(19)日本国特許庁（ＪＰ）　(12)公表特許公報（Ａ）　(11)特許出願公表番号
特表2002－503370
（P2002－503370A）
(43)公表日　平成14年１月29日(2002.1.29)

(51)Int.Cl.[7]	識別記号	ＦＩ	テーマコード（参考）
Ｇ０６Ｆ　9/38	３７０	Ｇ０６Ｆ　9/38	３７０Ｃ

審査請求　未請求　予備審査請求　有　（全134頁）

(21)出願番号　特願平11－501806
(86)(22)出願日　平成10年１月12日(1998.1.12)
(85)翻訳文提出日　平成11年11月25日(1999.11.25)
(86)国際出願番号　ＰＣＴ／ＧＢ９８／００００８３
(87)国際公開番号　ＷＯ９８／５７２５６
(87)国際公開日　平成10年12月17日(1998.12.17)
(31)優先権主張番号　９７１２０４１．４
(32)優先日　平成９年６月10日(1997.6.10)
(33)優先権主張国　イギリス（ＧＢ）
(81)指定国　ＥＰ(ＡＴ，ＢＥ，ＣＨ，ＤＥ，ＤＫ，ＥＳ，ＦＩ，ＦＲ，ＧＢ，ＧＲ，ＩＥ，ＩＴ，ＬＵ，ＭＣ，ＮＬ，ＰＴ，ＳＥ)，ＣＮ，ＩＬ，ＪＰ，ＫＲ，ＲＵ

(71)出願人　エイアールエム　リミテッド
イギリス国シービー１　９エヌジェイ　ケンブリッジ、チェリー　ヒントン，フルパーン　ロード　110
(72)発明者　ヨーク、リチャード
イギリス国　ケンブリッジ、チェリイ　ヒントン、バイオレット　クロース　６
(72)発明者　シール、デビッド、ジェームズ
イギリス国　ケンブリッジ、ケインズ　ロード　68
(72)発明者　シムズ、ドミニク
イギリス国　ケンブリッジ、チェリイ　ヒントン、ウェッジウッド　ドライブ　７
(74)代理人　弁理士　浅村　皓　（外３名）

(54)【発明の名称】　コプロセッサ・データ・アクセス制御

(57)【要約】
中央処理装置コア（２）と、メモリ（８）と、コプロセッサ（４）を備えたデジタル信号処理システムが、コプロセッサ・メモリアクセス命令（例えば、ＬＤＣ，ＳＴＣ）を使用して動作する。これらのコプロセッサ・メモリアクセス命令内のアドレシングモード情報（Ｐ，Ｕ，Ｗ，Ｏｆｆｓｅｔ）は、中央処理装置コア（２）によって使用されるアドレシングモードを制御するだけではなく、コプロセッサ（４）が、何個のデータワードが転送されるのかを決めるのにも使用され、コプロセッサ（４）は、適当な瞬間に転送を終了することができる。転送されるワードの個数を事前に知っておくことは、同期ＤＲＡＭと一緒に使用されるバスシステムにおいても有利である。命令内のオフセットフィールドは、特定の命令が実行されるときに、中央処理装置コア（２）によって提供される値に対する変更を指定するのに使用され、また、転送されるワード数を指定するのにも使用することができる。このような装置は、デジタル信号処理動作のような規則的なデータアレイを介する動作にうまく適合する。オフセットフィールドが使用されない場

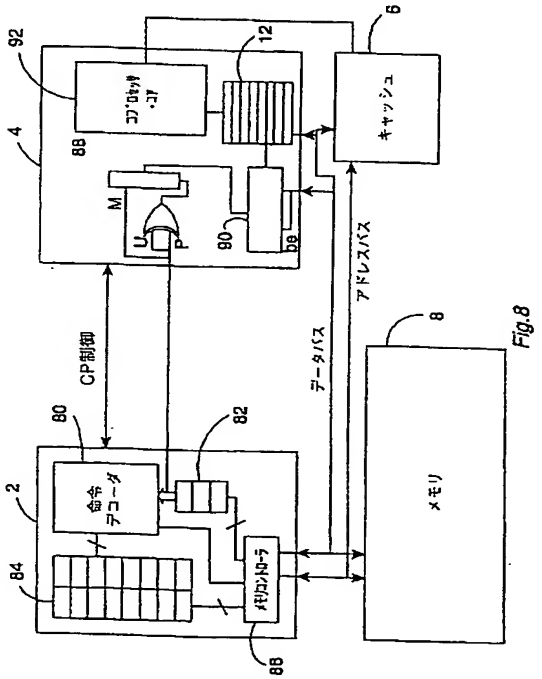


Fig.8

(19)日本国特許庁（ＪＰ）　(12)公表特許公報（Ａ）　(11)特許出願公表番号
特表2002－503370
（P2002－503370A）
(43)公表日　平成14年１月29日(2002.1.29)

(51)Int.Cl.⁷	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　9/38	３７０	Ｇ０６Ｆ　9/38	３７０Ｃ

審査請求　未請求　予備審査請求　有　（全134頁）

(21)出願番号　特願平11－501806
(86)(22)出願日　平成10年１月12日(1998.1.12)
(85)翻訳文提出日　平成11年11月25日(1999.11.25)
(86)国際出願番号　ＰＣＴ／ＧＢ９８／００００３
(87)国際公開番号　ＷＯ９８／５７２５６
(87)国際公開日　平成10年12月17日(1998.12.17)
(31)優先権主張番号　９７１２０４１．４
(32)優先日　平成９年６月10日(1997.6.10)
(33)優先権主張国　イギリス（ＧＢ）
(81)指定国　ＥＰ(ＡＴ，ＢＥ，ＣＨ，ＤＥ，ＤＫ，ＥＳ，ＦＩ，ＦＲ，ＧＢ，ＧＲ，ＩＥ，ＩＴ，ＬＵ，ＭＣ，ＮＬ，ＰＴ，ＳＥ)，ＣＮ，ＩＬ，ＪＰ，ＫＲ，ＲＵ

(71)出願人　エイアールエム　リミテッド
イギリス国シーピー１　９エヌジェイ　ケンブリッジ、チェリー　ヒントン，フルバーン　ロード　110
(72)発明者　ヨーク、リチャード
イギリス国　ケンブリッジ、チェリイ　ヒントン、バイオレット　クロース　６
(72)発明者　シール、デビッド、ジェームズ
イギリス国　ケンブリッジ、ケインズ　ロード　68
(72)発明者　シムズ、ドミニク
イギリス国　ケンブリッジ、チェリイ　ヒントン、ウェッジウッド　ドライブ　７
(74)代理人　弁理士　浅村　皓　（外３名）

(54)【発明の名称】　コプロセッサ・データ・アクセス制御

(57)【要約】
中央処理装置コア（２）と、メモリ（８）と、コプロセッサ（４）を備えたデジタル信号処理システムが、コプロセッサ・メモリアクセス命令（例えば、ＬＤＣ，ＳＴＣ）を使用して動作する。これらのコプロセッサ・メモリアクセス命令内のアドレシングモード情報（Ｐ，Ｕ，Ｗ，Ｏｆｆｓｅｔ）は、中央処理装置コア（２）によって使用されるアドレシングモードを制御するだけではなく、コプロセッサ（４）が、何個のデータワードが転送されるのかを決めるのにも使用され、コプロセッサ（４）は、適当な瞬間に転送を終了することができる。転送されるワードの個数を事前に知っておくことは、同期ＤＲＡＭと一緒に使用されるバスシステムにおいても有利である。命令内のオフセットフィールドは、特定の命令が実行されるときに、中央処理装置コア（２）によって提供される値に対する変更を指定するのに使用され、また、転送されるワード数を指定するのにも使用することができる。このような装置は、デジタル信号処理動作のような規則的なデータアレイを介する動作にうまく適合する。オフセットフィールドが使用されない場

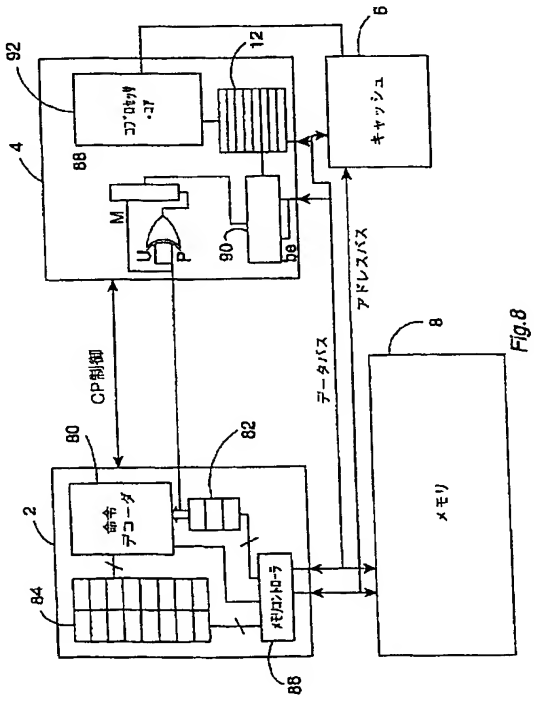


Fig.8

【特許請求の範囲】

１．データ処理装置であって、

コプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行う中央処理装置と、

前記中央処理装置に結合され、データワードを保持するメモリと、

前記中央処理装置と前記メモリに結合されたコプロセッサであって、コプロセッサにより処理されるメモリ内のデータワードをアドレスするのに、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用することを特徴とするコプロセッサと

を備えるデータ処理装置であって、

少なくとも１つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレシングモードのどれを使用するかを制御するアドレシングモード情報を含み、前記コプロセッサは、前記アドレシングモード情報の少なくとも一部を使用して、前記少なくとも１つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御する

データ処理装置。

２．請求項１に記載の装置であって、前記少なくとも１つのコプロセッサ・メモリアクセス命令が、アドレス値を持つ前記中央処理装置内のレジスタを参照し、前記アドレスモード情報がオフセットフィールドを含み、そこでは、アクセスされるべき前記メモリ内の開始アドレスが、前記少なくとも１つのコプロセッサ・メモリアクセス命令の実行に際して、前記アドレス値と前記オフセット値の少なくとも１つから決まることを特徴とする装置。

３．請求項２に記載の装置であって、前記アドレス値への前記変更が、最終アドレス値を生成し、それが前記レジスタに書き戻されることを特徴とする装置。

４．請求項２と３のいずれかに記載の装置であって、前記オフセットフィールドの少なくとも一部が、前記コプロセッサにより使用されて、前記メモリと前記コプロセッサとの間で転送されるデータの個数を制御することを特徴とする装置。

5. 請求項４に記載の装置であって、前記アドレシングモード情報が１つ又は２つ以上のフラグを持ち、それが、前記複数のアドレシングモードのどれが使用されているかを制御し、且つ、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを決める際に、前記オフセットフィールドが前記コプロセッサによって使用されるべきかどうかを制御することを特徴とする装置。

6. 請求項５に記載の装置であって、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを決める際に、前記オフセットフィールドが前記コプロセッサによって使用されない場合、固定数のデータワードが前記メモリと前記コプロセッサとの間で転送されることを特徴とする装置。

7. 請求項５に記載の装置であって、前記レジスタがアドレス値Ｒｎを記憶し、データワードの長さがＷＬバイトであり、前記オフセット値がＭであり、前記１つ又は２つ以上のフラグが、３つ以上の値ビットを備え、それらが、前記少なくとも１つのコプロセッサ・メモリアクセス命令を選択し、以下の１つに従って動作することを特徴とする装置。

	転送開始アドレス値	アドレスレジスタ内の最終値	転送されるデータワード個数
(i)	Rn	Rn-(WL*M)	（固定値）
(ii)	Rn	Rn	M
(iii)	Rn	Rn+(WL*M)	M
(iv)	Rn-(WL*M)	Rn	M
(v)	Rn-(WL*M)	Rn-(WL*M)	M
(vi)	Rn+(WL*M)	Rn	（固定値）
(vii)	Rn+(WL*M)	Rn+(WL*M)	（固定値）

8. 請求項７に記載の装置であって、前記フラグが、

（ｉ）前記アドレス値が元々前記レジスタに記憶されていたものであるか、前記オフセットフィールドによって変更されたアドレス値であるかを指定するフラグ

ビットＰと、

（ｉｉ）前記変更が、元々レジスタに記憶されていた値から、前記オフセットフィールドで指定された値の加算又は減算のどちらであるかを指定するフラグビットＵと、

（ｉｉｉ）前記アドレスレジスタ内の前記最終値が、前記レジスタに書き戻されるべきかどうかを指定するフラグビットＷと

を備えていることを特徴とする装置。

9. 請求項８に記載の装置であって、前記コプロセッサがＰ＝ＥＯＲ　Ｕを求めて、データワードを１個又はＭ個転送すべきかを決定することを特徴とする装置。

10. 請求項８に記載の装置であって、前記レジスタが前記中央処理装置のプログラムカウンタレジスタＰＣであり、前記コプロセッサが、転送されるデータワードが１個かＭ個かを決めるために、Ｐ＝ＥＯＲ（ＵＯＲ（レジスタはＰＣ））を求めることを特徴とする装置。

11. 以上の請求項のいずれかに記載の装置であって、前記中央処理装置及び前記コプロセッサが、デジタル信号処理を行い、前記メモリと前記コプロセッサとの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備えることを特徴とする装置。

12. 請求項６及び請求項７乃至１１のいずれかに記載の装置であって、前記固定数のデータワードが単一のデータワードを含む事を特徴とする装置。

13. データ処理方法であって、

中央処理装置によりコプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行うステップと、

データワードを、前記中央処理装置に結合されたメモリで保持するステップと、前記中央処理装置と前記メモリに結合されたコプロセッサによって、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用することによって、処理されるメモリ内のデータワードをアドレスするステップと

を備えるデータ処理方法であって、

少なくとも１つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレシングモードのどれを使用するかを制御するアドレシングモード情報を含み、前記コプロセッサは、前記アドレシングモード情報の少なくとも一部を使用して、前記少なくとも１つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御する

データ処理方法。

【発明の詳細な説明】

コプロセッサ・データ・アクセス制御

本発明は、データ処理システムの分野に関する。特に、中央処理装置、メモリ、コプロセッサを備えたデータ処理システムに関するもので、中央処理装置とコプロセッサにより一緒に実行される命令の制御下で、データワードがメモリとコプロセッサの間でやりとりされるシステムに関する。

中央処理装置、メモリ、コプロセッサ、例えば英国ケンブリッジのアドヴァンスト（Ａｄｖａｎｃｅｄ）ＲＩＳＣマシン株式会社によって製造されたＡＲＭマイクロプロセッサを備えたコプロセッサを備えた演算システムは公知である。そのような公知のシステムにおいて、中央処理装置は、（例えば、コプロセッサにロードする、あるいは記憶する）コプロセッサのメモリアクセス命令を実行する。これらの命令は、メモリへの供給のための適切なアドレスデータを生成し、コプロセッサがメモリと直接的にデータワード（データ転送の単位）を交換できるようにコプロセッサを準備させる。中央処理装置により開始アドレス情報がメモリに提供されると、中央処理装置を介したり、中央処理装置に記憶される、コプロセッサへ直接的にデータワードが渡されれば、最も効率が良い。コプロセッサへのそのような直接的転送を行えれば、コプロセッサがデータ転送の終了を制御することによって、中央処理装置を変更する必要なしに、異なったワード数を持つ異なったコプロセッサを中央処理装置に付属させることができる。データ転送の終了を制御するためには、中央処理装置上で実行される命令に応答して、コプロセッサは、何個のデータワードが転送されるべきかを決定しなければならない。

１つの可能性として、各命令が中央処理装置上で単一のワードだけの転送を行うということができる。ただし、これはデータメモリバンド幅の使用においても、また、コード側や命令メモリバンド幅においても非常に能率が悪いので、バーストモード転送が所望される。即ち、開始アドレスがメモリに提供され、メモリが隣接する一連のメモリ位置からデータワードを返すのが好ましい。バーストモード転送の場合、効率は良くなるが、困難も生じる。即ち、何個のデータワードが

転送されるのかをコプロセッサが決定することによって、コプロセッサが転送を終了するのに必要な制御を行えるようにしなければならない。

（例えば、ＡＲＭ浮動小数点アクセラレータユニットにおけるように）、中央処理装置上で実行する命令内にビットフィールドを割り当てることが公知であり、そこでは、ビットフィールド命令がコプロセッサに渡され、転送されるべきデータワードの個数をコプロセッサに指定する。しかしながら中央処理装置上で実行される命令内で使えるビットスペースには限界があり、もし、命令内のビットがコプロセッサへのデータワードの個数を渡すことに専念すると、命令内の他のフィールド用に使えるビットスペースを制限することになる。それは例えば、命令実行に続く中央処理装置内のアドレスポインタの変更のような、データ転送に関する他のパラメータを指定するのに使用されるかもしれない。

本発明の一面は、データ処理用装置であって、

コプロセッサメモリアクセス命令を含むデータ処理動作を行う中央処理装置命令を実行する中央処理装置と、

この中央処理装置に結合され、データワードを保持するメモリと、

中央処理装置とメモリに結合されたコプロセッサであって、このコプロセッサにより処理されるべきメモリ内のデータワードの指定が、中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用して行われることを特徴とするコプロセッサと

を備え、

少なくとも１つのコプロセッサメモリアクセス命令が、前記複数のアドレシングモードのどれが中央処理装置によって使用されるかを制御するアドレシングモード情報を含み、コプロセッサは、前記アドレシングモード情報の少なくとも一部を使用して、前記少なくとも１つのコプロセッサメモリアクセス命令に応答してメモリとコプロセッサの間で何個のデータワードが転送させるべきかを制御する。

本発明は、中央処理装置が複数のアドレシングモードのどれを使用するかを制御するのに使用する（レジスタ番号や即値定数を含むことのある）ビットフィールドは、コプロセッサに、転送されるべきデータワードの個数を指定するのにも

使用される（命令内の他のフィールドあるいは制御レジスタに書き込まれた値のような他の因子と結合されている可能性がある）ことがある。例えば、これまでにわかっているところ、多くの場合、中央処理装置が、転送及び／又はアドレスポインタへの変更に使用されアドレスを制御するのに使用するビットフィールド情報は、コプロセッサに転送されるデータワードの個数と関係づけられるので、このビットフィールドは、コプロセッサによっても中央処理装置によっても読むことができる。コプロセッサメモリアクセス命令内で同じビットフィールドを重複して使用することにより、このようなコプロセッサメモリアクセス命令内のビットスペースを他の目的に開放することになる。更に、現在わかっているところでは、大多数の場合、転送されるべきデータワードは、少数のカテゴリに分類されるので、コプロセッサに転送されるべき個数のデータワード用に専用ビットフィールドを提供するフルフレキシビリティは、本発明を使用することによって得られるコード密度と性能の改善に負ける。

中央処理装置は、いくつかの異なる方法でアドレシングを制御できるのに対して、本発明の好ましい実施の形態においては、前記少なくとも１つのコプロセッサメモリアクセス命令は、アドレス値を保持する中央処理装置内のレジスタを参照し、前記アドレスモード情報は、オフセットフィールドを備え、アクセスされるべきメモリ内の開始アドレスの決定は、前記少なくとも１つのコプロセッサメモリアクセス命令の実行により、前記アドレス値と前記オフセット値の少なくとも１つから行われる。

中央処理装置のレジスタ内に保持されるアドレスポインタのそのような使用は、多大なフレキシビリティを提供し、命令内のオフセットと結合されて、アプリケーションのコーディングを簡単にする。

コプロセッサとデータアレイを介して動作することが所望される場合、本発明の好ましい実施の形態では、前記アドレス値に行われた変更が、最終アドレス値を生成し、それを前記レジスタへ記憶しなおすことができるようにする。

前記オフセットフィールドの少なくとも一部が前記コプロセッサに使用されて、データワードが何個、前記メモリと前記コプロセッサの間で転送されるかを制御できることは、非常に便利であることがわかった。

このような装置は、メモリ内に保持されているデータ上にデータ処理操作をするためにコプロセッサが使用される実際の状況のかなりの部分の要求を満たすことができる。

また、アドレシングモード情報が１つ以上のフラグを備え、前記複数のアドレシングモードのうちのどれが使用されるか、また、何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを決定するのに前記オフセットフィールドが使用されるか否かを制御できることは有利である。

アドレシングモード情報内の他のフラグビットに依存してオフセットフィールドを選択的に使用することによって、転送されるデータワードの個数をコプロセッサが制御できる方法のオプションの数を増加させ、それにより、コプロセッサメモリアクセス命令内のビット空間を余分に必要とすることなしに、実際の状況の、より高い部分の要求に合わせることができる。

高い割合の所望のタイプの動作を成就するには、好ましい実施の形態は、次のようなものでなければならない。即ち、前記コプロセッサが、前記オフセットを使用せずに、前記メモリと前記コプロセッサの間で転送されたデータワードの個数を決定する場合、メモリとコプロセッサの間では、固定数のワードが転送される。

コプロセッサを制御するコプロセッサメモリアクセス命令モードの相補セットは、次のようなものである。即ち、前記レジスタがアドレスＲｎを記憶し、データワードがＷＬバイトの長さで、前記オフセット値がＭであり、前記１つ以上のフラグが、３つ以上の値ビットを備え、それらが、前記少なくとも１つのコプロセッサアクセス命令を選択し、以下の１つに従って操作する。

	転送開始 アドレス値	アドレスレジスタ 内の最終値	転送される ワード個数
(i)	Rn	Rn-(WL*M)	(固定値)
(ii)	Rn	Rn	M
(iii)	Rn	Rn+(WL*M)	M
(iv)	Rn-(WL*M)	Rn	M
(v)	Rn-(WL*M)	Rn-(WL*M)	M
(vi)	Rn+(WL*M)	Rn	(固定値)
(vii)	Rn+(WL*M)	Rn+(WL*M)	(固定値)

コプロセッサメモリアクセス命令は、以下のようなフラグを備えると有利である。

（ｉ）前記開始アドレス値が、元々前記レジスタに記憶されていたアドレス値であるか、前記オフセットフィールドによって指定される変更されたアドレス値であるかを指定するフラブビットＰ、

（ｉｉ）前記変更が、前記レジスタに元々記憶されていた値からオフセットフィールドによって指定される値を加算したものか減算したものであるかを指定するフラグビットＵ、

（ｉｉｉ）前記アドレスレジスタ内の前記最終値が、前記レジスタに記憶されなおすべきか否かを指定するフラグビットＷ。

このようなフラグセットを使用してコプロセッサは、高速で簡単な動作制御を行うことのできる構成となる。即ち、Ｐ　ＥＯＲ　Ｕを求めることによって、１個又はＭ個のデータワードが転送されるべきかを決定することができる。

更に、ベースレジスタがＡＲＭプログラムカウンタレジスタ（ＰＣ又はＲ１５）であれば、転送されるワード数１を生成することができる。この場合、単一ワード転送を決定するロジックが、Ｐ　ＥＯＲ（Ｕ又は（ベースレジスタがＰＣ））に変更される。

前述のコプロセッサと、中央処理装置及びメモリとの相互作用を制御する特性は多数の異なるフィールドに使用できる（例えば、浮動小数点コプロセッサ）場合、データアクセスは、次のように、比較的規則的である。即ち、前記中央処理装置と前記コプロセッサがデジタル信号処理を行い、前記メモリと前記コプロセッサの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備える。

本発明を他の面から見ると、本発明は以下のステップを備えるデータ処理方法を提供する。即ち、

中央処理装置により、コプロセッサメモリアクセス命令を含む中央処理装置命令を実行してデータ処理操作を行うステップと、

前記中央処理装置に結合されたメモリにおいてデータワードを保持するステップと、

前記中央処理装置に結合されたコプロセッサにより処理される前記メモリ内のデータワードをアドレスするステップであって、前記メモリは、前記中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用するステップと

を備え、

前記コプロセッサメモリアクセス命令の少なくとも１つは、前記複数のアドレシングモードのどれを中央処理装置が使用して前記メモリをアクセスするかを制御するアドレシングモード情報を含み、前記コプロセッサは、前記少なくとも１つのコプロセッサメモリアクセス命令に応答して、前記アドレシングモード情報の少なくとも一部を使用して何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを制御する方法である。

本発明の実施の形態について、以下に添付図面を参照しながら、例を示す。

図１は、デジタル信号処理装置のハイレベルの構成を示し、

図２は、コプロセッサの入力バッファとレジスタ構成を示し、

図３は、コプロセッサ内のデータパス（ｄａｔａｐａｔｈ）を示し、

図４は、レジスタからハイ又はローのビットを読むためのマルチプレクシング回路を示し、

図５は、好ましい実施の形態におけるコプロセッサにより使用されるレジスタ・リマッピング・ロジックを示すブロック図であり、

図６は、図５に示されたレジスタ・リマッピング・ロジックを更に詳しく示し

図７は、ブロック・フィルタ・アルゴリズムを示す表であり、

図８は、中央処理装置と、メモリと、コプロセッサアクセスメモリ命令を実行するためのコプロセッサとを備えたシステムを模式的に示し、

図９は、図８のシステムの動作に対応するフロー・ダイアグラムである。

以下の説明において、セクション１では、中央処理装置と、メモリと、高速デジタル信号処理能力を持つコプロセッサとを備えたシステムについて説明する。セクション２は、セクション１のシステムの変形について述べるもので、そこでは、コプロセッサメモリアクセス命令が、転送されるデータワードの個数のコプロセッサによる制御をより簡単にすべく変更されている。

セクション１

以下に説明するシステムは、デジタル信号処理（ＤＳＰ）に関する。ＤＳＰは、いろいろな形態を取ることができるが、典型的には、大量のデータの高速（実時間）処理を必要とする処理である。このデータは、典型的には、アナログの物理的信号である。ＤＳＰの好例として、デジタル移動電話に使用されるものがある。そこでは、無線信号が送受信され、アナログ音声信号から、及びアナログ音声信号へのデコーディング及びエンコーディング（典型的には、畳み込み(convolution)、変換、相関の操作を使用）が必要となる。また、他の例として、ディスクヘッドからの信号が処理されてヘッド・トラッキング制御が行われるディスク・ドライバ・コントローラが挙げられる。

上記のような文脈において、マイクロプロセッサ・コア（ここでは、英国、ケンブリッジのアドヴァンスト・ＲＩＳＣ・マシン・リミテッドにより設計されたマイクロプロセッサの範囲からのＡＲＭコア）上でのデジタル信号処理システムの説明をする。マイクロプロセッサとコプロセッサ・アーキテクチャとの間のインターフェースは、それ自体が、ＤＳＰ機能を提供すべく具体的構成を持つ。以下の説明において、マイクロプロセッサ・コアはＡＲＭ、コプロセッサはピッコ

ロ(Piccolo)とする。ＡＲＭとピッコロは、典型的には、他の構成要素（たとえば、チップ上のＤＲＡＭ、ＲＯＭ，Ｄ／Ａコンバータ、Ａ／Ｄコンバータ）をＡＳＩＣの部分として含む単一の集積回路として製造される。

ピッコロは、ＡＲＭのコプロセッサであるから、ＡＲＭ命令の集合の一部を実行する。ＡＲＭコプロセッサ命令により、（Load Coprocessor，ＬＤＣ and Store Coprocessor，ＳＴＣ命令を使用して）ＡＲＭがピッコロとメモリーの間でデータをやり取りさせ、また、（move to coprocessor，ＭＣＲ，及び、move from coprocessor，ＭＣＲ命令を使用して）ＡＲＭがＡＲＭレジスタをピッコロとやり取りすることができる。ある見方をすれば、ＡＲＭとピッコロの相互作用は、ＡＲＭＧＡピッコロのデータに対して強力なアドレス生成器として作用し、ピッコロの方は、大量のデータを実時間で扱う必要のあるＤＳＰ操作を自由に行うことによって、対応の実時間結果を生み出すことである。

図１は、ＡＲＭ２がピッコロ４に制御信号を発行して、データワードをピッコロ４に対して送信させ、またデータワードをピッコロ４から転送させる様子を示す。命令キャッシュ６は、ピッコロ４にとって必要なピッコロプログラム命令ワードを記憶する。単一のＤＲＡＭメモリ８は、ＡＲＭ２とピッコロ４の両方にとって必要なすべてのデータ及び命令ワードを記憶する。ＡＲＭ２は、メモリ８へのアドレッシング（addressing）及びすべてのデータ転送の制御に責任がある。単一のメモリ８、及び１セットのデータバスとアドレスバスから成る構成は、複数のメモリと高い帯域幅のバスを必要とする典型的ＤＳＰアプローチに比較して、構成が簡単であり、費用も易い。

ピッコロは、命令キャッシュ６からの第２の命令ストリーム（デジタル信号処理プログラム命令ワード）を実行し、これにより、ピッコロのデータパスが制御される。これらの命令は、デジタル信号処理方式操作、例えば、Multiply－Accumulate（乗算－累算）、及び制御フロー命令、例えば、ゼロ・オーバーヘッド・ループ命令を含む。これらの命令は、ピッコロのレジスタ１０（図２を参照）に保持されているデータを操作する。このデータは、前もって、ＡＲＭ２によってメモリ８から転送されたものである。複数の命令が命令キャッシュ６からストリ

ームとして出され、命令キャッシュ６が、データバスを、完全な支配下に置く。小型ピッコロ命令キャッシュ６は、１行当たり１６ワードの４行で、直接

マップされたキャッシュ（６４個の命令）となる。導入の方法によっては、命令キャッシュをもっと大きくしてもよい。

このように、２つのタスクが独立的に走る。ＡＲＭがデータをロードして、ピッコロがそれを処理する。これにより、１６ビット・データ上で単一サイクル・データ処理が維持される。ピッコロの持つデータ入力メカニズム（図２に示される）により、ＡＲＭは、シーケンシャル・データを先に取り込み、そのデータがピッコロに必要になるより先にロードする。ピッコロは、ロードされたデータにどのような順序でもアクセスすることができ、古いデータが最後に使用されると、自動的にそのレジスタを再び満たす（すべての命令はソースオペランド１つにつき、ソースレジスタを再充填すべきであることを示す１ビットを持つ）。この入力メカニズムは、リオーダ(reorder)バッファと呼ばれ、入力バッファ１２を備える。ピッコロにロードされる（以下に示すＬＤＣまたはＭＣＲを介して）すべての値には、その値の目的地がどのレジスタであるかを示すタグＲｎが付いている。タグＲｎは、入力バッファ内のデータワードの側に記憶される。あるレジスタがレジスタ選択回路１４を介してアクセスされ、命令がデータレジスタの再充填を指定すると、そのレジスタは、信号Ｅによって「空き」の印がつく。すると、レジスタは、自動的に、再充填制御回路１６によって、その入力バッファ１２内でそのレジスタに向けられた最も早くロードされた最古の値を充填される。リオーダ・バッファは８つのタグ付き値を保持する。入力バッファ１２の形式は、ＦＩＦＯと似ているが、キーの中央からデータワードを抽出することができ、その後で、遅くに記憶されたワードが渡され、その空き場所を埋める。従って、入力から最も遠いデータワードが最古であり、入力バッファ１２が正しいタグＲｎを持つ２つのデータワードを保持する時は、その最古のデータワードを使用して、どちらのデータワードでレジスタを再充填すべきかを決定することができる。

ピッコロは、図３に示されたように、データを出力バッファ１８（ＦＩＦＯ）

に記憶させて出力する。データはＦＩＦＯにシーケンシャルに書き込まれ、ＡＲＭによって同じ順序でメモリ８に読み出される。出力バッファ１８は、８つの３２ビットの値を保持する。

ピッコロは、コプロセッサ・インターフェース（図１のＣＰ制御信号）を介してＡＲＭと接続する。ＡＲＭコプロセッサ命令の実行に際して、ピッコロは、それを実行するか、あるいは、ピッコロがその命令を実行できるようになるまでＡＲＭを待たせるか、あるいは命令実行を拒否することができる。最後の場合、ＡＲＭは、未定義命令例外とする。

ピッコロが実行する最も普通のコプロセッサ命令はＬＤＣとＳＴＣであり、これらは、それぞれデータワードをデータバスを介してメモリ８へ、及びメモリ８からロードし（ＬＤＣ）、記憶させ（ＳＴＣ）、ＡＲＭがすべてのアドレスを生成する。リオーダ・バッファにデータをロードし、出力バッファ１８からのデータを記憶するのもこれらの命令である。ピッコロは、入力リオーダ・バッファにデータをロードするのに充分な場所がなければＡＲＭをＬＤＣのままにし、また出力バッファに記憶すべき充分なデータがなければＡＲＭをＳＴＣのままにする。ピッコロは、また、ＡＲＭ／コプロセッサ・レジスタ転送を行って、ＡＲＭがピッコロの特定の(special)レジスタにアクセスできるようにする。

ピッコロは、それ自身の命令はメモリから取り込み、図３に示されたピッコロのデータパスを制御し、リオーダ・バッファからレジスタへ、またレジスタから出力バッファ１８へデータを転送する。これらの命令を行うピッコロの演算ユニットは、乗算／加算回路２０を有し、これが乗算、加算、減算、乗算・累算、論理操作、シフト、及び回転を行う。また、データパスには累算／退出（decumulate）回路２２と、縮尺(scale)／飽和（saturate）回路２４とが備わっている。

ピッコロ命令は、最初にメモリから命令キャッシュ６にロードされ、そこへピッコロがアクセスし、主記憶にアクセスバックする必要がない。

メモリがアボート(abort)した場合、ピッコロはそれを修復することができない。従って、ピッコロを仮想メモリシステムで使用する場合、すべてのピッコロのデータは、ピッコロのタスクの始めから終わりまで、物理的メモリになければ

ならない。このことは、ピッコロのタスクの実時間性、例えば実時間ＤＳＰを考えれば、大した問題ではない。メモリ・アボートが起きると、ピッコロは停止して状態レジスタＳ２にフラグをセットする。

図３は、ピッコロの全体のデータパス機能を示す。レジスタ・バンク１０は、３つの読み出しポートと２つの書き込みポートを使用する。１つの書き込みポート（Ｌポート）は、リオーダ・レジスタからレジスタを再充填するのに使用される。出力バッファ１８は、ＡＬＵ結果バス２６から直接的に更新され、出力バッファ１８からの出力は、ＡＲＭプログラム制御の支配下にある。ＡＲＭコプロセッサ・インターフェースは、ＬＤＣ（Load coprocessor）命令をリオーダ・バッファに行い、出力バッファ１８からＳＴＣ(Store Coprocessor)命令を行い、また、レジスタバンク１０上にＭＣＲとＭＲＣ(Move ＡＲＭ register to／from ＣＰ register)を行う。

残りのレジスタ・ポートは、ＡＬＵに使用される。読み出しポート（Ａ及びＢ）は、入力を乗算／加算回路２０に駆動し、Ｃ読み出しポートは、累算（accumulate）／退出（decumulate）回路２２入力の駆動に使用される。残りの書き込みポートＷは、結果をレジスタバンク１０に戻すのに使用される。

乗算器２０は、符号付き又は符号無し１６ｘ１６の乗算を行い、必要により４８ビット累算を伴うこともできる。スケーラー（scaler）ユニット２４は、０から３１までの即値算術又は論理シフト右を提供することができ、その後、必要により飽和を行うことができる。シフタ(shifter)及び論理ユニット２０は、各周期でシフト又は論理操作を行うことができる。

ピッコロは、Ｄ０－Ｄ１５又はＡ０－Ａ３，Ｘ０－Ｘ３，Ｙ０－Ｙ３，Ｚ０－Ｚ３という名のついた１６個の汎用レジスタを持つ。最初の４つのレジスタ（Ａ０－Ａ３）は、累算用で、４８ビットの幅があり、余分な１６ビットが、多数の連続的計算の間にオーバーフローが生じないためのガードを提供する。残りのレジスタは３２ビットの幅である。

ピッコロのレジスタは各々２つの独立した１６ビットの値を含むものとして扱うことができる。ビット０からビット１５までが下半分、ビット１６からビット

３１までが上半分を含む。命令は、ソースオペランドとして各レジスタのどちらかの半分の１６ビットを指定することができ、あるいは、全体の３２ビットレジスタを指定することもできる。

また、ピッコロは、飽和演算に対する備えもある。乗算、加算、減算命令の変量は、結果が目的レジスタのサイズより大きい場合、飽和結果を提供する。目的レジスタが４８ビットのアキュムレータであれば、値は３２ビットで飽和される（つまり、４８ビットの値を飽和させる方法はない）。４８ビットのレジスタにはオーバーフローの検出がない。これは手頃な制限である。というのは、オーバーフローを起こすには、少なくとも６５５３６乗算累算命令が必要であるから。

各ピッコロのレジスタは、「空き」（Ｅフラグ、図２参照）であるか、１つの値を含む（レジスタの半分だけが空きになることはない）。初期状態では、すべてのレジスタが空きの印がついている。各周期で、ピッコロは再充填制御回路１６によって、空きレジスタの１つを、入力リオーダ・バッファからの値で埋める。あるいは、レジスタにＡＬＵからの値が書き込まれている場合は、「空き」ではない。もし、レジスタにＡＬＵからの書き込みがあり、これと同時に、リオーダ・バッファからのレジスタに入れられる値が控えている場合は、結果は未定義である。空きレジスタに読み出しが行われれば、ピッコロの実行ユニットはとまってしまう。

入力リオーダ・バッファ（ＲＯＢ）は、コプロセッサ・インターフェースとピッコロのレジスタ・バンクとの間にある。データがＲＯＢにロードされる時は、ＡＲＭコプロセッサが転送する。ＲＯＢは、多数の３２ビットの値を含み、それぞれ値の目的地となるピッコロ・レジスタを示すタグを持っている。タグは、また、そのデータが３２ビットレジスタの全体に転送されるのか、あるいは３２ビット中の下の１６ビットだけに転送されるべきかも示す。データがレジスタ全体に転送される場合は、そのエントリーの下の１６ビットは目的レジスタの下半分に転送され、上の１６ビットはレジスタの上半分に転送される（目的レジスタが４８ビット・アキュムレータの場合は、符号が拡張される）。データの目的地がレジスタの下半分だけ（いわゆるハーフ・レジスタ）の場合、下の１６ビットが

先に転送される。

レジスタのタグは常に物理的目的レジスタを示し、レジスタのリマッピングが行われることはない（レジスタのリマッピングについては、以下を参照）。

各周期で、ピッコロは、次のように、データ・エントリをＲＯＢからレジスタ・バンクへ転送しようとする。

－　ＲＯＢの各エントリが検査され、タグが空きレジスタと比較され、エントリの一部又は全部からレジスタへ転送が可能かどうかが決定される。

－　転送可能なエントリの集合から、最古のエントリが選択され、そのデータがレジスタバンクへ転送される。

－　このeントリのタグが更新されてエントリを空きにする。エントリの一部だけが転送された場合は、転送された部分だけが空きの印になる。

例えば、目的レジスタが完全に空きであり、選択されたＲＯＢエントリが含むデータが１つの全体レジスタ用であれば、３２ビット全体が転送され、そのエントリは空きの印になる。目的レジスタの下半分が空きであり、ＲＯＢの含むデータがレジスタの下半分用であれば、ＲＯＢエントリの下の１６ビットが目的レジスタの下半分へ転送され、ＲＯＢの下半分が空きの印になる。

どのエントリでも、上の１６ビットと下の１６ビットは、それぞれ独立に転送することができる。レジスタバンクに転送できるデータを含むエントリが皆無の場合、その周期では、転送は行われない。下の表は、目的ＲＯＢエントリと目的レジスタ状態のあらゆる可能な組み合わせを示す。

	目的，Rn，状態		
目的 ROB エントリ状態	空き	下半分空き	上半分空き
レジスタ全体 両半分有効	Rn.h <- entry.h Rn.l <- entry.l エントリ「空き」	Rn.l <- entry.l エントリ下半分 「空き」	Rn.h <- entry.h エントリ上半分 「空き」
レジスタ全体 上半分有効	Rn.h <- entry.h エントリ「空き」		Rn.h <- entry.h エントリ「空き」
レジスタ全体 下半分有効	Rn.l <- entry.l エントリ「空き」	Rn.l <- entry.l エントリ「空き」	
レジスタ半分 両半分有効	Rn.l <- entry.l エントリ下半分 「空き」	Rn.l <- entry.l エントリ下半分 「空き」	
レジスタ半分 上半分有効	Rn.l <- entry.h エントリ「空き」	Rn.l <- entry.h エントリ「空き」	

以上をまとめると、１つのレジスタの２つの半分は、互いに独立に、ＲＯＢから充填することができる。ＲＯＢ内のデータは、レジスタ全体用に印が付けられるか、あるいはレジスタの下半分用の２つの１６ビットの値としての印が付く。

データをＲＯＢにロードするにはＡＲＭコプロセッサ命令が使用される。ＲＯＢにおいてデータが印を付けられる方法は、転送に使用されたＡＲＭコプロセッサ命令がどれであったかによる。ＲＯＢにデータを充填するのに使用できるＡＲＭ命令には以下のものがある。

ＬＤＰ {<cond>} <16/32>　　<dest>, [Rn] { ! } ,#<size>

ＬＤＰ {<cond>} <16/32>W　　<dest>, <wrap>, [Rn] { ! } ,#<size>

ＬＤＰ {<cond>} 16U　　<bank>, [Rn] { ! }

ＭＰＲ {<cond>}　　<dest>, Rn

ＭＲＰ {<cond>}　　<dest>, Rn

ＲＯＢの構成には、以下のＡＲＭ命令が提供される。

ＬＤＰＡ<bank list>

最初の３つは、ＬＤＣ命令としてアセンブルされ、ＭＰＲとＭＲＰは、ＭＣＲ命令として、ＬＤＰＡはＣＤＰ命令としてアセンブルされる。

上記＜dest＞は、ピッコロのレジスタ（Ａ０－Ｚ３）を示し、ＲｎはＡＲＭレジスタを示し、＜size＞は４の乗数（ゼロを除く）となる定数としてのバイト数であり、＜wrap＞は、定数（１、２、４、８）を示す。{}によって囲まれた領域は、オプションである。転送がリオーダ・バッファへ当てはまるようにするために、＜size＞は最大で３２である。多くの場合、＜size＞は、この制限より小さくしてデッドロックを避ける。＜１６／３２＞領域は、ロードされるデータが１６ビット・データとして扱われ、endianess特定動作（以下を参照）を行うべきか、あるいは３２ビットデータであるかを示す。

注１：以下の説明において、ＬＤＰまたはＬＤＰＷに言及する場合、これらの命令の１６ビット用変種と３２ビット用変種の両方を含むものとする。

注２：１つのワード（word）は、メモリからの３２ビットの固まりであり、それは、１６ビットのデータ項目２つから成るか、あるいは３２ビットのデータ項目１つからなる。

ＬＤＰ命令は、多数のデータ項目をフル・レジスタ用として転送する。この命令は、メモリ内のアドレスＲｎから＜size＞／４ワードをロードし、それらをＲＯＢに挿入する。転送することのできるワード数は以下のように制限される。

－　＜size＞の量は、４の非ゼロ倍数でなければならず、

－　＜size＞は、特定の導入についてＲＯＢのサイズ以下でなければならない（第１版では８ワード、その後の版では、それ以下にならない保証があること）。

転送される最初のデータ項目は目的地が＜dest＞のタグを付け、第２のデータ

項目は、＜dest＞＋１というようになる（Ｚ３からＡ０まではラッピング（wrapping））。もし“！”が指定された場合は、その後、レジスタＲｎが＜size＞によって１つずつ増加される。

ＬＤＰ１６の変種が使用された場合は、エンダイアン（endian）特定動作が２つの１６ビットのハーフワードに行われて、それらがメモリシステムから戻される時には３２ビットデータ項目とする。より詳しくは、以下のBig Endian及びLittle Endianサポートを参照せよ。

ＬＤＰＷ命令は、多数のデータ項目をレジスタのセットに転送する。最初に転送されるデータ項目には＜dest＞のタグが付き、次は＜dest＞＋１のタグが付き、以下同様。＜wrap＞転送が起きると、次に転送される項目は、＜dest＞用となり、以下同様。＜wrap＞の量は、ハーフワードの量で指定される。

ＬＤＰＷには、次の制限がある。

－　＜size＞の量は、４の非ゼロ倍でなければならず、

－　＜size＞は、特定の導入についてＲＯＢのサイズ以下でなければならず（第１版では８ワード、その後の版では、それ以下にならない保証がある）、

－　＜dest＞は、｛Ａ０，Ｘ０，Ｙ０，Ｚ０｝のいずれか１つでよく、

－　＜wrap＞は、ＬＤＰ３２Ｗについては｛２，４，８｝のいずれかの個数のハーフワードであり、ＬＤＰ１６Ｗについては｛１，２，４，８｝のいずれかの個数のハーフワードであり、

－　＜size＞の量は、２＊＜wrap＞より大きくなければならない。さもないと、ラッピングは起きず、代わりにＬＤＰ命令が使用される。

たとえば、次の命令

ＬＤＰ３２Ｗ　　Ｘ０，２，〔Ｒ０〕！，＃８

は、２つのワードをＲＯＢにロードし、その目的地をフル・レジスタＸ０とする。Ｒ０は、８増加する。次の命令

ＬＤＰ３２ｗ　　Ｘ０，４，〔Ｒ０〕，＃１６

は、４つのワードをＲＯＢにロードし、それらの目的地をＸ０，Ｘ１，Ｘ０，Ｘ１（この順序で）とする。Ｒ０は影響されない。

ＬＳＰ１６Ｗに対しては、＜wrap＞は、１、２、４、又は８として指定できる

。

１のラップが指定されると、すべてのデータのタグの目的地が、目的レジスタの下半分＜dest＞．１．となる。これは、ハーフ・レジスタの場合である。

例えば、次の命令

ＬＤＰ１６Ｗ　　Ｘ０，１，〔Ｒ０〕！，＃８

は、２つのワードをＲＯＢにロードし、それらを１６ビットデータとして目的地をＸ０．１とする。Ｒ０は８増加される。次の命令

ＬＤＰ１６Ｗ　　Ｘ０，４，〔Ｒ０〕，＃１６

は、ＬＤＰ３２Ｗの例と同様に挙動するが、ただし、エンダイアン特定動作は、メモリから戻されるデータ上に行われる。

ＬＤＰ命令のすべての使用されないエンコーディングは、将来の拡張用に取っておくことができる。

ＬＤＰ１６Ｕ命令は、非ワード揃え（non-word aligned）１６ビットデータの効率良い転送をサポートする。ＬＤＰ１６Ｕサポートはレジスタ Ｄ４－Ｄ１５（Ｘ，Ｙ，Ｚバンク）になされる。ＬＤＰ１６Ｕサポートは、レジスタ３２ビットワードのデータ１つ（２つの１６ビットデータ項目を含む）をメモリからピッコロへ転送することになる。ピッコロは、このデータの下の１６ビットを捨て、上の１６ビットを保持レジスタに記憶する。Ｘ，Ｙ，Ｚバンク用の保持レジスタがある。バンクの保持レジスタが通報されると（primed）と、データの目的地がそのバンク内のレジスタであれば、ＬＤＰ｛ｗ｝命令の挙動が変化する。ＲＯＢにロードされたデータは、ＬＤＰ命令によって転送されつつあるデータの下の１６ビットと保持レジスタとの連結によって形成される。転送されつつある上の１６ビットは、保持レジスタに入れられる。

entry <- data.l¦ holding__register

holding__register<-data.h

このモードの動作は、ＬＤＰＡ命令によって打ち切られるまで続く。保持レジスタは、目的レジスタのタグもサイズも記録しない。これらの性質は、次のdata．１．の値を提供する命令から得られる。

メモリシステムから戻されたデータには、常にエンダイアン特定挙動が起きる可能性がある。ＬＤＰ１６Ｕと同等の非１６ビットはない。というのは、３２ビットデータ項目はすべてメモリにおいてワード揃えされるからである。

ＬＤＰＡ命令は、ＬＤＰ１６Ｕ命令によって開始された操作の非整列(unaligned)モードを取り止めるのに使用される。非整列モードは、バンクＸ，Ｙ，Ｚ上で独立に切ることができる。例えば、次の命令

ＬＤＰＡ　　｛Ｘ，Ｙ｝

は、バンクＸ，Ｙ上で非整列モードを打ち切る。これらのバンクの保持レジスタ内のデータは、捨てられる。

非整列モードにないバンク上でＬＤＰＡを実行することは可能であり、そのバンクは整列モードのままである。

ＭＰＲ命令は、ＡＲＭレジスタＲｎの内容をＲＯＢに入れ、ピッコロレジスタ＜dest＞に向けられる。目的レジスタ＜dest＞は、Ａ０－Ｚ３の範囲のフルレジスタならどれでもよい。例えば、次の命令

ＭＰＲ　　Ｘ０，Ｒ３

は、Ｒ３の内容をＲＯＢに移し、そのデータをフルレジスタＸ０用とする。

データがＡＲＭからピッコロに転送される時にエンダイアネス(endianess)特定挙動が生じることがない。というのは、ＡＲＭは、内部的に、あまりエンダイアンではないからである。

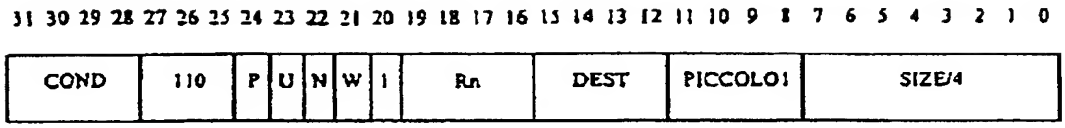
ＭＰＲＷ命令は、ＡＲＭレジスタＲｎの内容をＲＯＢに入れ、それを、１６ビットピッコロレジスタ＜dest＞．１．向けの２つの１６ビットデータ項目とする。＜dest＞についての制限は、ＬＤＰＷ命令の場合と同じである（つまり、Ｚ０，Ｘ０，Ｙ０，Ｚ０）。例えば、次の命令

ＭＰＲＷ　　Ｘ０，Ｒ３

は、Ｒ３の内容をＲＯＢに移し、Ｘ０．１．向けの２つの１６ビット量のデータとする。尚、１でラップするＬＤＰ１６Ｗの場合、３２ビットレジスタの下半分だけが目的地となり得る。

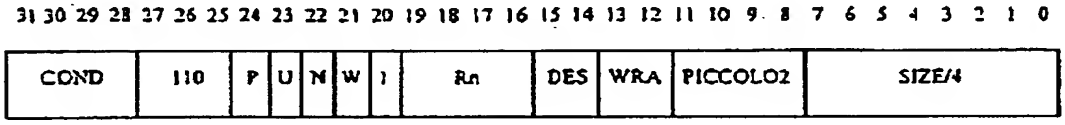
ＭＰＲについては、データに対してエンダイアネス特定操作は何も行われない

ＬＤＰは、次のようにエンコードされる。



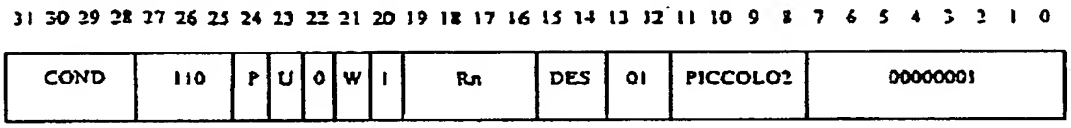
ここで、ＰＩＣＣＯＬＯ１は、ピッコロの最初のコプロセッサの番号（現在８）である。ＮビットがＬＤＰ３２（１）とＬＤＰ１６（０）との間の選択を行う。

ＬＤＰＷは、次のようにエンコードされる。



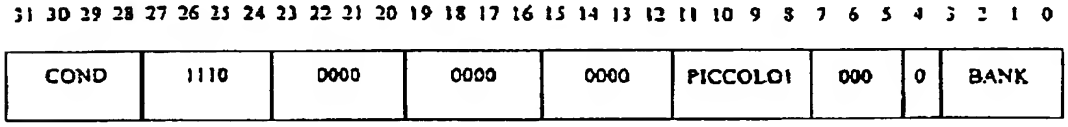
ここで、ＤＥＳＴは、目的レジスタＡ０，Ｘ０，Ｙ０，Ｚ０に対する０－３であり、ＷＲＡＰは、１、２、４、８の値のラップに対して０－３である。ＰＩＣＣＯＬＯ２は、ピッコロの第２のコプロセッサ番号（現在９）である。Ｎビットが、ＬＤＰ３２（１）とＬＤＰ１６（０）との間の選択を行う。

ＬＤＰ１６Ｕは、次のようにエンコードされる。



ここで、ＤＥＳＴは、目的バンクＸ，Ｙ，Ｚに対する１－３である。

ＬＤＰＡは、次のようにエンコードされる。



ここで、ＢＡＮＫ〔３：０〕は、バンクごとの非整列モードを打ち切るのに使用

される。ＢＡＮＫ〔１〕がセットされると、バンクＸ上の非整列モードが打ち切られる。ＢＡＮＫ〔２〕及びＢＡＮＫ〔３〕がセットされれば、それぞれバンクＹ，Ｚ上の非整列モードが打ち切られる。尚、これはＣＤＰ操作である。

ＭＰＲは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0	1	0	0	DEST				Rn				PICCOLO1				000			1	0000			

ＭＰＲＷは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0	1	0	0	DEST				00				Rn				PICCOLO2				000			1	0000	

ここで、ＤＥＳＴは、目的レジスタＸ０，Ｙ０，Ｚ０に対する１－３である。

出力ＦＩＦＯは、３２ビットの値を８つまで保持することができる。これらは、次の（ＡＲＭ）オペコード(opcodes)の１つを使用して、ピッコロから転送される。

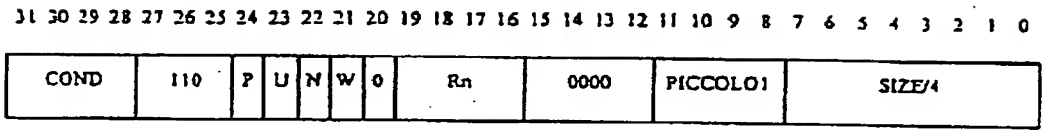
ＳＴＰ {<cond>} <16/32>　　[Rn] { ! } ,#<size>

ＭＲＰ　　Rn

最初のは、＜size＞／４ワードを出力ＦＩＦＯから、ＡＲＭレジスタＲｎによって与えられるアドレスへ退避する。“！”があれば、Ｒｎを指示する。デッドロックを避けるために、＜size＞は、出力ＦＩＦＯのサイズ（この導入例では８エントリ）以下でなければならない。ＳＴＰ１６の変種が使用された場合は、メモリシステムから戻されるデータにエンダイアン特定挙動が生じる可能性がある。

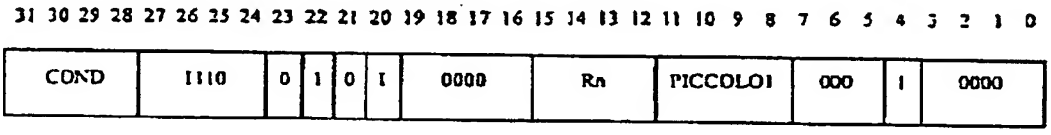
ＭＲＰ命令は、出力ＦＩＦＯから１つのワードを除去し、それをＡＲＭレジスタＲｎに入れる。ＭＲＰと同様に、このデータには、エンダイアン特定操作が適用されることはない。

ＳＴＰ用のＡＲＭエンコーディングは以下の通り。



ここで、Ｎは、ＳＴＰ３２（１）とＳＴＰ１６（０）との間の選択を行う。Ｐ，Ｕ，Ｗビットの定義については、ＡＲＭデータシートを参照せよ。

ＭＲＰ用のＡＲＭエンコーディングは以下の通り。



ピッコロ命令セットは、内部的にはエンダイアン操作がほとんどないと仮定している。例えば、３２ビットレジスタに、複数の１６ビット・ハーフとしてアクセスする場合、下半分がビット１５から０を占めるとする。ピッコロは、大きなエンダイアン・メモリ又は周辺機器(peripherals)のあるシステムで動作することになるので、１６ビットでパックされたデータを正しくロードできるようにしなければならない。

ピッコロ（つまり、ＤＳＰが採用されたコプロセッサ）は、ＡＲＭ（例えば、英国、ケンブリッジのアドヴァンストＲＩＳＣマシンズ・リミテッドによって製造されたＡＲＭ７マイクロプロセッサ）のように、プログラマがプログラム可能周辺機器で制御できるであろう‘ＢＩＧＥＮＤ’構成ピンを持っている。ピッコロは、このピンを使用して入力リオーダ・バッファ及び出力ＦＩＦＯを構成する。

ＡＲＭがパック１６ビットデータをリオーダ(reorder)・バッファにロードする時は、そのことを、ＬＤＰ命令の１６ビット形式を使用して示さなければならない。この情報は‘ＢＩＧＥＮＤ’構成入力の状態と組み合わされて、データを保持ラッチへ入れ且つリオーダ・バッファを適当な順序にする。特にbig endianモードの時は、保持レジスタはロードされたワードの下１６ビットを記憶し、次のロードの上１６ビットと対（ペア）にされる。保持レジスタの内容は、常に、

リオーダ・バッファへ転送されたワードの下１６ビットで終わる。

出力ＦＩＦＯは、パックされた１６ビット又は３２ビットデータを含むことができる。プログラマは、ＳＴＰ命令の正確な形式を使用して、１６ビットデータがデータバスの正しい半分に提供されていることをピッコロが確認できるようにしなければならない。big endianとして構成されている場合、ＳＴＰの１６ビット形式が使用されると、上１６ビットハーフ及び下１６ビットハーフが交換される。

ピッコロは、ＡＲＭからしかアクセスできないプライベート・レジスタを４tu持っている。これらは、Ｓ０－Ｓ２と呼ばれる。これらにアクセスできるのは、ＭＲＣ命令とＭＣＲ命令だけである。オペコードは以下の通り。

ＭＰＳＲ　　Ｓｎ，Ｒｍ

ＭＲＰＳ　　Ｒｍ，Ｓｎ

これらのオペコードは、ＡＲＭレジスタＲｍとプライベート・レジスタＳｎとの間で３２ビット値を転送する。それらは、ＡＲＭにおいて、コプロセッサ・レジスタ転送としてエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				001			L	Sn						Rm				PICCOLO				000		I	0000			

ここで、Ｌは、ＭＰＳＲなら０、ＭＲＰＳなら１である。

レジスタＳ０は、ピッコロの一意なＩＤ及び改定コードを含む。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Implementor								Architecture				Part Number																Revision			

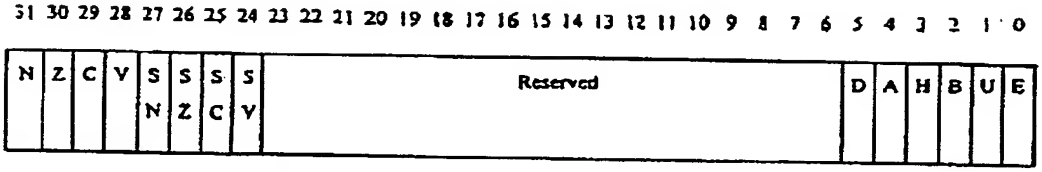
〔３：０〕ビットは、プロセッサの改定番号を含む。

〔１５：４〕ビットは、２進符号化された１０進フォーマットの３桁部分の番号（ピッコロなら、０ｘ５００）を含む。

〔２３：１６〕ビットは、アーキテクチャ版数を含む。０ｘ００＝第１版

〔３１：２４〕ビットは、導入者の商標のＡＳＣＩＩコードを含む。０ｘ４１＝Ａ＝ＡＲＭ　Ｌｔｄ．

レジスタＳ１は、ピッコロの状態レジスタである。



一次状件コードフラグ（Ｎ，Ｚ，Ｃ，Ｖ）

二次状件コードフラグ（ＳＮ，ＳＺ，ＳＣ，ＳＶ）

Ｅビット：ピッコロは、ＡＲＭによってディスエーブルされ、中止した。

Ｕビット：ピッコロは、ＵＮＤＥＦＩＮＥＤ（未定義）命令に出会って、中止した。

Ｂビット：ピッコロは、ＢＲＥＡＫＰＯＩＮＴ（区切点）に出会って、中止した。

Ｈビット：ピッコロは、ＨＡＬＴ（中止）命令に出会って、中止した。

Ａビット：ピッコロは、メモリ・アボート（ロード、ストア、又はピッコロ命令）によって、中止した。

Ｄビット：ピッコロは、デッドロック条件を検出し、中止した（以下を参照）。

レジスタＳ２はピッコロプログラム・カウンタである。

プログラム・カウンタに書き込みすると、ピッコロはそのアドレスで（中止状態であれば中止状態のまま）プログラムの実行を始める。プログラム・カウンタはリセットされた時、未定義である。というのは、プログラム・カウンタへの書

き込みによって、ピッコロは常にスタートされるからである。

実行中、ピッコロは命令の実行及びコプロセッサ・インターフェースの状態を次のようにモニタする。

－　ピッコロは、レジスタ再充填されるのを、あるいは出力ＦＩＦＯが使えるエントリを持つのを、待つ態勢に入った。

－　ＲＯＢ内のスペースが不十分であるか、出力ＦＩＦＯ内の事項(items)が不十分であるかの理由で、コプロセッサ・インターフェースがビジー待ち状態(busy－waiting)にある。

これらの両方の条件が検出されると、ピッコロは、その状態レジスタにＤビットをセットし、中止し、ＡＲＭコプロセッサの命令を拒絶し、ＡＲＭは未定義命令トラップにはまる。

このデッドロック状態の検出により、少なくともプログラマにこのような条件が生じたことを知らせ、また失敗の正確な点（位置）を知らせることができるシステムが構成される。プログラマは、ＡＲＭとピッコロのプログラム・カウンタとレジスタを読めばよい。尚、強調しておくが、デッドロックが生じるのは、間違ったプログラムあるいはピッコロの状態を変造するシステム部分がある場合だけである。デッドロックは、データが少なすぎることや「オーバーロード」によって生じることはない。

ＡＲＭからピッコロを制御するのに使用できるいくつかの操作があり、それらはＣＤＰ命令によって提供される。これらのＣＤＰ命令は、ＡＲＭが優先状態にある時に受け付けられる。そうでないと、ピッコロはＣＤＰ命令を拒絶し、ＡＲＭは未定義命令トラップにはまる。以下の操作が使用可能である。

－　Reset（リセット）

－　Enter State Access Mode（状態アクセスモードに入る）

－　Enable（イネーブル）

－　Disable（ディスエーブル）

ピッコロは、ＰＲＥＳＥＴ命令によってソフトウェア内でプリセットされる。

ＰＲＥＳＥＴ　　；Clear Piccolo's state

（ピッコロの状態をクリアする）

この命令は、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0000				0000				0000				PICCOLO1				000			0	0000			

この命令が実行されると、次のことが生じる。

－　すべてのレジスタが空き（再充填の態勢）の印になる。

－　入力ＲＯＢがクリアされる。

－　出力ＦＩＦＯがクリアされる。

－　ループ・カウンタがリセットされる。

－　ピッコロは中止状態に入る（そしてＳ２のＨビットがセットされる）。

ＰＲＥＳＥＴ命令の実行には、いくつかのサイクル（この実施の形態では、２から３）が必要である。実行されている間に、以下のピッコロ上で実行されるべきＡＲＭコプロセッサ命令がビジー待ちになる。

状態アクセスモードにおいて、ピッコロの状態は、ＳＴＣ及びＬＤＣ命令（以下のＡＲＭからのピッコロ状態アクセスについての説明を参照）を使って退避され復元される。状態アクセスモードに入るには、ＰＳＴＡＴＥ命令がまず実行されなければならない。

ＰＳＴＡＴＥ；　Enter State Access Mode

（状態アクセスモードに入る）

この命令は次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0001				0000				0000				PICCOLO1				000			0	0000			

実行されると、ＰＳＴＡＴＥ命令は、

－　ピッコロを中止し（すでに中止されているのでなければ）、Ｅビットをピッ

コロの状態レジスタにセットする。

－　ピッコロを状態アクセスモードに構成する。

ＰＳＴＡＴＥ命令の実行が終わるまでにはいくつかのサイクルがある。というのは、ピッコロの命令パイプラインは中止する前に汲み出されなければならないからである。実行中、ピッコロ上で実行される次のＡＲＭコプロセッサ命令がビジー待ちになる。

ＰＥＮＡＢＬＥ及びＰＤＩＳＡＢＬＥ命令は、高速コンテキスト切替えに使用される。ピッコロがディスエーブルされると、専用レジスタ０と１だけが（ＩＤレシスタ、状態レジスタ）アクセス可能となり、それも優先モードからだけである。これ以外の状態へアクセスすると、またユーザモードからアクセスすると、ＡＲＭ未定義命令例外が生じる。ピッコロをディスエーブルすると、実行が中止される。ピッコロは、実行を中止すると、状態レジスタにＥビットをセットして応答する。

ピッコロをイネーブルするには、ＰＥＮＡＢＬＥ命令を実行する。

ＰＥＮＡＢＬＥ；　Enable Piccolo

この命令は次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0010				0000				0000				PICCOLO1				000			0	0000			

ピッコロをディスエーブルするには、ＰＤＩＳＡＢＬＥ命令を実行する。

ＰＤＩＳＡＢＬＥ；　Disable Piccolo

この命令は次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110				0011				0000				0000				PICCOLO1				000			0	0000			

この命令が実行されると、次のことが生じる。

－　ピッコロの命令パイプラインが空になる(drain)。

－　ピッコロは中止して、状態レジスタにＨビットをセットする。

－　このセクションは、ピッコロのデータパスを制御するピッコロ命令セット（集合）に言及する。各命令は３２ビットの長さである。これらの命令は、ピッコロ命令キャッシュから読み出される。

命令セットのデコードは、非常に直線的である。最初６ビット（２６から３１）が主要オペコードを与え、２２から２５までが、いくつかの特定の命令のためのマイナーなオペコードを提供する。灰色の影となっているコードは、現在未使用のものであり、拡張用として使える（それらは現時点で指示された値を含んでいなければならない）。

１１の主要命令クラスがある。これは、いくつかのサブクラスのデコードを簡単にするため、命令にファイルされた主要オペコードに完全に対応するものではない。

3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	9	8	7	6	5	4	3	2	1	0
0	OPC						F	SD	DEST				SI	RI	SRC1				SRC2											
1	000			OPC			F	SD	DEST				SI	RI	SRC1				SRC2											
1	001			0	OP		F	SD	DEST				SI	RI	SRC1				SRC2											
1	0011																													
1	010			OPC			F	SD	DEST				SI	RI	SRC1				SRC2_SHIFT											
1	011			00			F	SD	DEST				SI	RI	SRC1				SRC2_SEL								COND			
1	011			01																										
1	011			1	OP		F	SD	DEST				SI	RI	SRC1				SRC2_SEL								COND			
1	10	0		OP	Sa		F	SD	DEST				AI	RI	SRC1				SRC2_MULA											
1	10	1			0																									
1	10	1		OP	1		F	SD	DEST				AI	RI	SRC1		0	A0	R2	SRC2_REG				SCALE						
1	110																													
1	11100						F	SD	DEST				IMMEDIATE_15																	
1	11101																													
1	11110					0	RFIELD_4					0	RI	SRC1						#INSTRUCTIONS_8											
1	11110					1	RFIELD_4				#LOOPS_13									#INSTRUCTION_8											
1	11111					0	OPC				REGISTER_LIST_16									SCALE											
1	11111					100					IMMEDIATE_16									COND											
1	11111					101					PARAMETERS_21																			
1	11111					11	OP																							

上の表にある命令には、以下の名前がついている。

Standard Data Operation　（標準データ操作）

Logical Operation　（論理操作）

Conditional Add/Subtract　（条件付加算／減算）

Undefined　（未定義）

Shifts　（シフト）

Select　（選択）

Undefined　（未定義）

Parallel Select　（並列選択）

Multiply Accumulate　（乗算累算）

Undefined　（未定義）

Multiply Double　（乗算ダブル）

Undefined　（未定義）

Move Signed Immediate　（符号付即値移動）

Undefined　（未定義）

Repeat　（反覆）

Repeat　（反覆）

Register List Operation　（レジスタ・シフト操作）

Branch　（ブランチ）

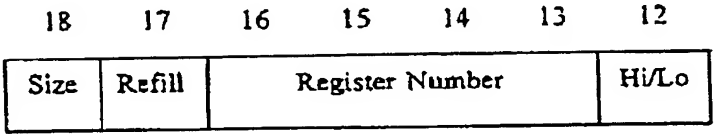
Renaming Parameter Move　（リネーム・パラメータ移動）

Halt/Break　（中止／中断）

命令の各クラスのフォーマットは、次のセクションに詳しく述べてある。ソース及び目的オペランド領域は、ほとんどの命令において共通であり、レジスタ・リマッピングと同様、別のセクションに述べてある。

ほとんどの命令は２つのソースオペランドSource１，Source２を必要とする。

Source１（ＳＲＣ１）オペランドは、次の７ビット・フォーマットを持つ。



この領域の要素は、次の意味を持つ。

－　Size－読み出すオペランドのサイズを示す（１＝３２ビット、０＝１６ビット）。

－　Refill－レジスタが読み出された後、空きの印になり、ＲＯＢから再充填できることを示す。

－　Register Number－３２ビット、１６ビットレジスタのどっちのレジスタを読み出すべきべきかエンコードする。

－　Ｈｉ／Ｌｏ－１６ビット読み出しに対して、３２ビットレジスタのどちらの半分を読み出すべきかを示す。３２ビットオペランドに対してセットされた場合は、レジスタの２つの１６ビット半分が入れ換えられなければならないことを示す。

Size	Hi/Lo	Portion of Register Accessed
0	0	Low 16 bits
0	1	High 16 bits
1	0	Full 32 bits
1	1	Full 32 bits, halves swapped

レジスタのサイズは、レジスタ番号に接尾辞を付けることによってアセンブラによって特定される。下位１６ビットなら、．ｌ、上位１６ビットなら、．ｈ，３２ビットの上下の１６ビットを入れ換えるなら、．ｘ。

一般のソース２（ＳＣＲ２）は、次の３つの１２ビット・フォーマットの１つを持つ。

11	10	9	8	7	6	5	4	3	2	1	0
0	S2	R2	'Register Number				Hi/Lo	SCALE			
1	0	ROT	IMMED_8								
1	1	IMMED_6						SCALE			

図４は、選択されたレジスタの適切な半分をピッコロのデータパスにスイッチするためのＨｉ／Ｌｏビット及びSizeビットに応答するマルチプレクサ構成を示す。Sizeビットが１６ビットであれば、符号拡張回路がデータパスの高次ビットに適切な０または１を入れる。

最初のエンコーディングは、ソースをレジスタとして指定し、その領域は、ＳＣＲ１指定子（specifier）と同じエンコーディングを持つ。ＳＣＡＬＥ領域は、ＡＬＵの結果に適用されるべきスケールを指定する。

SCALE				Action
3	2	1	0	
0	0	0	0	ASR #0
0	0	0	1	ASR #1
0	0	1	0	ASR #2
0	0	1	1	ASR #3
0	1	0	0	ASR #4
0	1	0	1	RESERVED
0	1	1	0	ASR #6
0	1	1	1	ASL #1
1	0	0	0	ASR #8
1	0	0	1	ASR #16
1	0	1	0	ASR #10
1	0	1	1	RESERVED
1	1	0	0	ASR #12
1	1	0	1	ASR #13
1	1	1	0	ASR #14
1	1	1	1	ASR #15

８ビット即値(immediate)は、回転（rotate）エンコーディングによって、３２ビット即値を生成し、それが、８ビット値及び２ビット回転（rotate）によって表現される。次の表は、８ビット値ＸＹから生成される即値を示す。

ROT	IMMEDIATE
00	0x000000XY
01	0x0000XY00
10	0x00XY0000
11	0xXY000000

６ビット即値エンコーディングにより、６ビット符号無し即値（範囲０から６３）を、ＡＬＵの出力に提供されるスケールと共に使用することができる。

一般のソース２エンコーディングは、ほとんどの命令変種に共通である。この規則には例外が少しあり、それがソース２エンコーディングの限定されたサブセットをサポートするか、あるいは、それを少し変形させる。

- Select Instructions. （選択命令）
- Shift Instructions. （シフト命令）
- Parallel Operations. （並列操作）
- Multiply Accumulate Instructions. （乗算累算命令）
- Multiply Double Instructions. （乗算ダブル命令）

選択命令は、レジスタ又は１６ビット符号無し即値であるオペランドをサポートするだけである。スケールは無効である。それは、これらのビットは命令の条件領域によって使用されるからである。

	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_SEL	0	S2	R2	Register number				Hi/Lo	COND			
	1	1	IMMED_6						COND			

シフト命令は、１６ビットレジスタ又は５ビット符号無し即値である１から３１のオペランドをサポートするだけである。結果のスケールは無効である。

SRC2_SHIFT

11	10	9	8	7	6	5	4	3	2	1	0
0	0	R2	Register number				Hi/Lo	0	0	0	0
1	0	0	0	0	0	0	IMMED_5				

並列操作の場合、レジスタがオペランドのソースとして指定されていれば、３２ビット読み出しが行われなければならない。即値エンコーディングは、並列操作については、少し違った意味を持つ。これにより、即値は、３２ビットオペランドの１６ビット半分の両方に複製できる。並列操作には少し制限のある範囲のスケールが使用できる。

SRC2_PARALLEL

11	10	9	8	7	6	5	4	3	2	1	0
0	1	R2	Register number				Hi/Lo	SCALE_PAR			
1	0	ROT	IMMED_8								
1	1	IMMED_6						SCALE_PAR			

６ビット即値が使用された場合、常に、３２ビット量の半分の両方に複製される。８ビット即値が使用された場合は、それが複製されるのは、それが３２ビット量の上半分に回転されるべきであると回転（rotate）が示している時だけである。

ROT	IMMEDIATE
00	0x000000XY
01	0x0000XY00
10	0x00XY00XY
11	0xXY00XY00

並列選択操作にはスケールは無効である。スケール領域は、これらの命令では

０にセットされる。

乗算累算命令では、８ビット回転即値を指定することはできない。領域のビット１０は、どのアキュミュレータを使用すべきかを指定する部分となる。ソース２は、１６ビットオペランドとして意味される。

SRC2_MULA

11	10	9	8	7	6	5	4	3	2	1	0
0	A0	R2	Register number				Hi/Lo	SCALE			
1	A0	IMMED_6						SCALE			

乗算ダブル命令は、定数を使用することができない。１６ビットレジスタだけが指定できる。この領域のビット１０は、どのアキュミュレータを使用すべきかを指定する部分となる。

SRC2_MULD

11	10	9	8	7	6	5	4	3	2	1	0
0	A0	R2	Register number				Hi/Lo	SCALE			

命令のうちいくつかは、常に３２ビット操作（例えば、ＡＤＤＡＤＤ）を含み、その場合、サイズ・ビットは、１にセットされ、Ｈｉ／Ｌｏビットは、場合によっては３２ビットオペランドの２つの１６ビット半分を交換するのに使用することができる。また、いくつかの命令は、常に１６ビット操作（例えば、ＭＵＬ）を含み、サイズビットは０に設定されなければならない。Ｈｉ／Ｌｏビットは、レジスタのどちらの半分が使用されるかを選択する（見えないサイズビットは明らかなものと仮定する）。乗算・累算命令は、ソース・アキュミュレータと目的レジスタを独立に指定することができる。これらの命令においては、Sizeビットは、ソースアキュミュレータを指定するのに使用され、サイズビットは、命令タイプによって０と暗示される。

１６ビット値が（Ａ又はＢバスを介して）使用される場合、それは、自動的に

３２ビット量に符号拡張される。４８ビットレジスタが（Ａ又はＢバスを介して

）読みだされる場合、下の３２ビットだけがバスに現れる。それは、どの場合でも、ソース１、ソース２は、３２ビット値に変換されるからである。バスＣを使用する累算命令だけがアキュミュレータレジスタの４８ビット全部にアクセスすることができる。

再充填ビットがセットされていれば、レジスタは使用後、空きの印になり、普通の再充填メカニズムによってＲＯＢから再充填される（ＲＯＢについてのセクションを参照）。ピッコロは、再充填が行われる以前にソースオペランドとしてレジスタが再び使用されないかぎり、止まらない。再充填されたデータが有効になる前のサイクルの最小数（最善の場合で、データはＲＯＢの先頭で待っている）は、１か２である。従つて、再充填されたデータは、再充填要求の次の命令には使わない方が良い。もし、次の２つの命令上でオペランドの使用を避けることができるのであれば、その方が良い。というのは、これにより、より深いパイプライン導入上での性能損失を防ぐことになるから。

再充填ビットは、レジスタ番号に接尾辞“＾”を付けることによってアセンブラで指定される。空きの印のついたレジスタのセクションは、レジスタのオペランドに依存する。各レジスタの２つの半分は、独立に、再充填の印をつけることもできる（例えば、Ｘ０．１＾は、Ｘ０の下半分だけを再充填することになり、Ｘ０＾は、Ｘ０全体を再充填することになる）。４８ビットレジスタの上「半分」（ビット４７：１６）が再充填されると、１６ビットのデータがビット３１：１６に書き込まれ、ビット４７まで符号拡張される。

同じレジスタを２回再充填しようと（例えば、ＡＤＤ　Ｘ１，Ｘ０＾，Ｘ０＾）しても、再充填は１度しか行われない。アセンブラは、ＡＤＤ　Ｘ１，Ｘ０，Ｘ０＾という文法しか許可すべきではない。

レジスタ読み出しが、レジスタの再充填以前に試みられると、ピッコロは、レジスタが再充填されるまでまち状態で止まる。レジスタが再充填の印になると、レジスタは再充填の値が読まれる以前に更新され、その結果、ＵＮＰＲＥＤＩＣＡＴＢＬＥ（予想不可）となる（例えば、ＡＤＤ　Ｘ０，Ｘ０＾，Ｘ１は予想不可。なぜなら、Ｘ０については再充填の印であるから、Ｘ０と

Ｘ１の合計で埋めることになる）。

４ビットスケール領域は１４のスケールタイプをエンコードする。

－　ＡＳＲ　＃０，１，２，３，４，６，８，１０

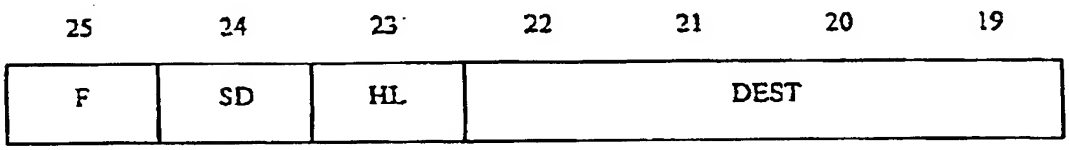
－　ＡＳＲ　＃１２から１６

－　ＬＳＬ　＃１

並列Ｍａｘ／Ｍｉｎ命令は、スケールを提供しないので、ソース２の６ビット定数変種は使用されない（アセンブラにより０にセットされる）。

ＲＥＰＥＡＴ命令内で、レジスタのリマッピングがサポートされ、ＲＥＰＥＡＴが、ループを解かないままレジスタの移動「窓」にアクセスすることができる。これについて、以下、詳しく説明する。

目的オペランドは、次の７ビットフォーマットを持つ。



この基本エンコーディングには１０の変種がある。

アセンブラ ニューモニック		25	24	23	22	21	20	19
Dx	1	0	1	0	Dx			
Dx^	2	1	1	0	Dx			
Dx.l	3	0	0	0	Dx			
Dx.l^	4	1	0	0	Dx			
Dx.h	5	0	0	1	Dx _			
Dx.h^	6	1	0	1	Dx			
未定義		0	1	1	0000			
.l（レジスタ書き戻しなし，16ビット）	7	1	1	1	0	0	00	
""（レジスタ書き戻しなし，32ビット）	8	1	1	1	0	1	00	
.l^（16-ビット）出力	9	1	1	1	1	0	00	
^（32-ビット）出力	10	1	1	1	1	1	00	

レジスタ番号（Ｄｘ）は１６のレジスタのどれがアドレスされているかを示す。Ｈｉ／ＬｏビットとSizeビットは、一緒になって、各３２ビットレジスタへ１６ビットレジスタのペアとしてアドレスする。Sizeビットは、どのようにしたら適切なフラグが、命令タイプに定義されているように、セットされるかを定義するもので、結果がレジスタバンク及び／又は出力ＦＩＦＯに書き込まれるか否かには関係しない。これにより、比較及び同様の命令の構成ができる。命令の累算クラスのある加算は、結果をレジスタに書き戻さなければならない。

エンコード	レジスタ書込	FIFO書込	V フラグ
1	レジスタ全体に書く	書込なし	32-ビット オーバーフロー
2	レジスタ全体に書く	32ビット書込	32-ビット オーバーフロー
3	ロー16ビットを Dx.1に書く	書込なし	16-ビット オーバーフロー
4	ロー16ビットを Dx.1に書く	ロー16ビットに 書込	16-ビット オーバーフロー
5	ロー16ビットを Dx.hに書く	書込なし	16-ビット オーバーフロー
6	ロー16ビットを Dx.hに書く	ロー16ビットに 書込	16-ビット オーバーフロー
7	書込なし	書込なし	16-ビット オーバーフロー
8	書込なし	書込なし	32-ビット オーバーフロー
9	書込なし	ロー16ビットに 書込	16-ビット オーバーフロー
10	書込なし	32ビット書込	32-ビット オーバーフロー

どの場合でも、レジスタへの書き戻し又は出力ＦＩＦＯへの挿入以前の操作の結果は、４８ビット量である。２つの場合がある。

書き込みが１６ビットならば、４８ビット量は、下の１６ビット〔１５：０〕を選択することによって、１６ビットに減る。命令が飽和すれば、値は、－２＾１５から２＾１５－１の範囲に飽和される。次に１６ビット値が指示されたレジスタに書き戻され、また、書き込みＦＩＦＯビットがセットされれば、出力ＦＩＦＯに書き戻される。出力ＦＩＦＯに書き込まれた場合、それは、次の１６ビット値が書き込まれるまで保持される。次の１６ビットが書き込まれると、それらの値はペアとなって、単一３２ビット値として出力ＦＩＦＯに入れられる。

３２ビットの書き込みならば、４８ビットは、下の３２ビット〔３１：０〕を選んで３２ビット量に減る。

３２ビット書き込みでも、４８ビット書き込みでも、命令が飽和すれば、４８ビット値は－２＾３１－１から２＾３１の範囲の３２ビット値に変換される。飽和すると、

－　アキュミュレーへの書き戻しが行われると、４８ビット全部が書き込まれる。

－　３２ビットレジスタへの書き戻しが行われると、ビット〔３１：０〕が書き込まれる。

－　出力ＦＩＦＯへの書き戻しが行われるた場合も、やはりビット〔３１：０〕が書き込まれる。

目的サイズは、レジスタ番号の後の．ｌまたは．ｈによって、アセンブラ内で指定される。レジスタへの書き戻しが全く行われない場合は、レジスタ番号は意味がなくなるので、目的レジスタを省略して、レジスタへの書き込み無しとするか、あるいは、＾を使って、出力ＦＩＦＯだけへの書き込みを指示する。例えば、ＳＵＢ，Ｘ０，Ｙ０は、ＣＭＰ　Ｘ０，Ｙ０と等価であり、ＡＤＤ＾，Ｘ０，Ｙ０は、Ｘ０＋Ｙ０の値を出力ＦＩＦＯに入れる。

出力ＦＩＦＯに値を入れる空きがない場合は、ピッコロは、空きができるまで待機する。

１６ビット値、例えば、ＡＤＤ　Ｘ０．ｈ＾，Ｘ１，Ｘ２が書き出されると、その値は、第２の１６ビット値が書かれるまでラッチされる。次にこれら２つの値は結合されて、３２ビット数として出力ＦＩＦＯに入れられる。最初に書き込まれる１６ビット値は、常に３２ビットワードの下半分に現れる。出力ＦＩＦＯに入れられたデータは、１６又は３２ビットデータとしての印がつき、endianessをbig endianシステム上で訂正することができる。

３２ビット値が２つの１６ビット書き込みの間に書き込まれると、その動作は未定義になる。

ＲＥＰＥＡＴ命令内で、レジスタ・リマッピングがサポートされ、ＲＥＰＥＡＴは、ループを解く（unroll）ことなしにレジスタの移動「窓」にアクセスす

ることができる。以下、これについて詳しく説明する。

本発明の好ましい実施の形態において、ＲＥＰＥＡＴ命令は、レジスタ・オペランドがループ内で特定される方法を変更するメカニズムを提供する。このメカニズムの下で、アクセスするレジスタは命令内のレジスタ・オペランドとレジスタバンクのオフセットの機能によって決定される。オフセットは、プログラム可能な方法で変更でき、各命令ループの最後で変更されるのが好ましい。このメカニズムは、Ｘ，Ｙ，Ｚバンク内にあるレジスタ上で独立に動作することができる。好ましい実施の形態では、この機能はΛバンク内のレジスタには使用できない。

論理レジスタ、物理レジスタという概念を使用することができる。命令オペランドは論理レジスタを参照し、これらは、特定のピッコロレジスタ１０を同定する物理レジスタ・レファレンスにマップされる。すべての操作は、再充填も含み、物理レジスタ上で動作する。レジスタ・リマッピングが生じるのは、ピッコロ命令ストリームサイドだけであり、ピッコロにロードされるデータは常に物理レジスタを目的とし、リマッピングは行われない。

リマッピングのメカニズムについて、以下、図５を参照して説明する。図５は、ピッコロ・コプロセッサ４の多数の内部構成要素を示すブロック図である。メモリからＡＲＭコア２によって検索されるデータ項目は、リオーダ・バッファ１２に入れられ、ピッコロレジスタ１０は、先に図２を参照した方法で、リオーダ・バッファ１２から再充填される。キャッシュ６に記憶されているピッコロの命令は、ピッコロ４内の命令デコーダ５０に渡されることによって、ピコロ・プロセッサ・コア５４に渡される前にデコードされる。ピッコロ・プロセッサ・コア５４は、先に図３を参照して述べた乗算器／加算器回路２０と、累算／退出回路２２と、スケール／飽和（saturate）回路２４とを備える。

命令デコーダ５０がＲＥＰＥＡＴ命令によって同定された命令ループの一部を構成する命令を扱っていて、且つ、そのＲＥＰＥＡＴ命令が多数のレジスタのリマッピングを行うことが必要であると指示した場合は、レジスタ・リマッピング論理５２が使用されて、必要なリマッピングが行われる。レジスタ・リマッピング論理５２は、命令デコーダ５０の一部であると考えて良い。ただし、当業者に

は明らかなように、レジスタ・リマッピング論理５２は、命令デコーダ５０に対して全く別のものとして提供されてもかまわない。

典型的な命令は、その命令にとって必要なデータ項目を含むレジスタを同定する１つまたは２つ以上のオペランドを備える。例えば、典型的な命令は、２つのソースオペランドと１つの目的ペランドを含むことができ、その命令が必要とするデータ項目を含む２つのレジスタと、その命令の結果を入れるべきレジスタを同定する。レジスタ・リマッピング論理５２は、命令デコーダ５０から命令のオペランドを受け取るが、それらは論理レジスタ・レファレンスを同定する。論理レジスタ・レファレンスに基づき、レジスタ・リマッピング論理は、物理レジスタのリマッピングをすべきかどうかを決定し、必要なら、物理レジスタ・レファレンスにリマッピングを適用する。また、リマッピングを適用すべきではないと決定された場合は、論理レジスタ・レファレンスが物理レジスタ・レファレンスとして提供される。リマッピングを行う好ましい方法については、後で、詳しく説明する。

レジスタ・リマッピング論理からの各出力物理レジスタ・レファレンスは、ピッコロ・プロセッサ・コア５４に渡されることによって、プロセッサ・コアが、物理レジスタ・レファレンスによって同定される特定のレジスタ１０内のデータ項目に命令を適用できるようにする。

好ましい実施の形態によるリマッピングのメカニズムによれば、レジスタの各バンクは、２つのセクション、つまりその中でレジスタがリマップされるセクションと、レジスタがリマッピング無しで元のレジスタ・レファレンスを保持するセクションとの２つのセクションに割ることができる。好ましい実施の形態において、リマップされたセクションは、リマップされているレジスタ・バンクの下から開始される。

このリマッピングのメカニズムは多数のパラメータを使用し、これらのパラメータについては、図６を参照しながら、詳細に説明する。図６は、様々なパラメータがレジスタ・リマッピング論理５２によっていかに使用されるかを示すブロック図である。尚、これらのパラメータは、リマップされているバンク内の点、

例えば、バンクの下からの相対的値を与えられている。

レジスタ・リマッピング論理５２は、２つの主要論理ブロック、つまりRemap（リマップ）ブロック５６とBase Update（ベース更新）ブロック５８とからなると考えることができる。レジスタ・リマッピング論理５２は、論理レジスタ・レファレンスに加えられるべきオフセット値を提供するベース・ポインタを使用する。このベース・ポインタの値は、ベース更新ブロック５８によってリマップ・ブロックに提供される。

ＢＡＳＥＳＴＡＲＴ信号を使用して、ベースポインタの初期値を定義することができる。例えば、典型的には、ゼロであるが、他の値を指定することもできる。このＢＡＳＥＳＴＡＲＴ信号は、ベース更新ブロック５８内のマルチプレクサ６０に渡される。命令ループの最初の繰り返しで、ＢＡＳＥＳＴＡＲＴ信号は、マルチプレクサ６０によって、記憶エレメント６６に渡され、ループのその後の繰り返しでは、次のベース・ポインタ値がマルチプレクサ６０によって記憶エレメント６６に渡される。

記憶エレメント６６の出力は、現在のベース・ポインタ値としてリマップ論理５６に渡され、またベース更新論理５８内の加算器６２の入力の１つにも渡される。加算器６２は、ベース・インクリメント値を提供するＢＡＳＥＩＮＣ信号を受け取る。加算器６２は、記憶エレメント６６によって供給される現在のベース・ポインタ値を、ＢＡＳＥＩＮＣ値分だけインクリメントし、結果をモジュロ回路６４へ渡すようになっている。

また、モジュロ回路は、ＢＡＳＥＷＲＡＰ値を受け取り、この値を加算器６２からの出力ベース・ポインタ信号と比較する。インクリメントされたベース・ポインタ値がＢＡＳＥＷＲＡＰ値以上であれば、その新しいベース・ポインタがラップラウンドされて、新しいオフセット値となる。モジュロ回路６４の出力は、記憶エレメント６６に記憶されるべき次のベース・ポインタ値となる。この出力はマルチプレクサ６０に提供され、そこから、記憶エレメント６６に提供される。

しかしながら、この次のベース・ポインタ値は、ＲＥＰＥＡＴ命令を管理する

ループ・ハードウェアからＢＡＳＥＵＰＤＡＴＥ信号を記憶エレメント６６が受け取らないうちは、記憶エレメント６６に記憶できない。ＢＡＳＥＵＰＤＡＴＥ信号は、ループ・ハードウェアによって周期的に生成され、例えば、命令ループが反復されるごとに、生成される。ＢＡＳＥＵＰＤＡＴＥ信号を記憶エレメント６６が受け取ると、記憶エレメントは、以前のベース・ポインタ値にマルチプレクサ６０から提供される次のベース・ポインタ値を上書きする。このように、リマップ論理５８に供給されるベース・ポインタ値は、新しいベース・ポインタ値に変わる。

レジスタバンクのリマップされたセクション内でアクセスされるべき物理レジスタは、命令のオペランド内に含まれる論理レジスタ・レファレンスに、ベース更新論理５８によって提供されるベース・ポインタ値を加えることによって決定される。この加算を行うのは加算器６８であり、その出力は、モジュロ回路７０に渡される。好ましい実施の形態において、モジュロ回路７０は、レジスタ・ラップ値を受け取り、加算器６８からの出力信号（論理レジスタ・レファレンスとベース・ポインタ値の和）がレジスタ・ラップ値を越えると、その結果へ、リマップされた領域の下でラップ（wrap）が行われる。モジュロ回路７０の出力は、マルチプレクサ７２に提供される。

ＲＥＧＣＯＵＮＴ値がリマップ・ブロック５６内の論理７４に提供され、リマップされるべきバンク内のレジスタの個数を同定する。論理７４は、このＲＥＧＣＯＵＮＴ値を論理レジスタ・レファレンスと比較し、比較の結果により、制御信号をマリチプレクサ７２に渡す。マルチプレクサ７２は、その２つの入力で、論理レジスタ・レファレンスとモジュロ回路７０からの出力（リマップされたレジスタ・レファレンス）を受け取る。本発明の好ましい実施の形態において、論理レジスタ・レファレンスがＲＥＧＣＯＵＮＴ値より小さければ、論理７４は、マルチプレクサ７２にリマップされたレジスタ・レファレンスを物理レジスタ・レファレンスとして出力させる。ただし、もし、論理レジスタ・レファレンスがＲＥＧＣＯＵＮＴ値以上であれば、論理７４は、マルチプレクサ７２に論理レジスタ・レファレンスを直接、物理レジスタ・レファレンスとして出力させる。

先に述べたように、好ましい実施の形態において、リマッピング・メカニズムを引き起こすのはＲＥＰＥＡＴ命令である。後で、より詳しく述べるように、ＲＥＰＥＡＴ命令は、ハードウェアで４つのゼロサイクルループを提供する。これらのハードウェア・ループは、図５に命令デコーダ５０の部分として図示されている。命令デコーダ５０がキャッシュ６から命令を要求する度に、キャッシュはその命令を命令デコーダに戻し、それにより、命令デコーダは、戻された命令がＲＥＰＥＡＴ命令であるかどうか判断する。もしそれであれば、ハードウェア・ループの１つが、そのＲＥＰＥＡＴ命令を扱うように構成される。

各繰り返し命令は、ループ内の命令の数と、ループを繰り返す回数（定数またはピッコロ・レジスタから読み出される）を指定する。２つのオペコードＲＥＰＥＡＴとＮＥＸＴがハードウェアループの定義用に提供され、ＮＥＸＴオペコードは単に区切りとして使用されるだけで、命令としてアセンブルはされない。ＲＥＰＥＡＴがループの頭に行き、ＮＥＸＴがループの最後を区切ることによって、アセンブラはループ・本体内の命令の数を数えることができる。好ましい実施の形態において、ＲＥＰＥＡＴ命令は、レジスタ・リマッピング論理５２が使用すべきＲＥＧＣＯＵＮＴ，ＢＡＳＥＩＮＣ，ＢＡＳＥＷＲＡＰ，ＲＥＧＷＲＡＰパラメータのようなリマッピング・パラメータを含むことができる。

レジスタ・リマッピング論理によって使用されるリマッピング・パラメータを記憶する多数のレジスタを提供することができる。これらのレジスタ内で、前もって定義されたリマッピング・パラメータの多数のセット（集合）を提供することができる一方、いくつかのレジスタはユーザ定義リマッピング・パラメータを記憶するために残される。ＲＥＰＥＡＴ命令と共に指定されたリマッピング・パラメータが、前もって定義されたリマッピング・パラメータの１つと等しい場合、適当なＲＥＰＥＡＴエンコーディングが使用され、これにより、マルチプレクサ等が適当なリマッピング・パラメータをレジスタから直接にレジスタ・リマッピング論理へ提供する。一方、リマッピング・パラメータが前もって定義されたリマッピング・パラメータのどれとも等しくない場合は、アセンブラがRemapping Parameter Move（ＲＭＯＶ）命令を生成する。これにより、ユーザ定義レジス

タ・リマッピング・パラメータの構成が可能となり、ＲＭＯＶ命令の後にＲＥＰＥＡＴ命令が続く。好ましくは、ユーザ定義リマッピング・パラメータは、ＲＭＯＶ命令によって、そのようなユーザ定義リマッピング・パラメータを記憶すべく残されていたレジスタに入れられ、マルチプレクサは、それらのレジスタの内容をレジスタ・リマッピング論理に渡すようプログラムされる。

好ましい実施の形態において、ＲＥＧＣＯＵＮＴ，ＢＡＳＥＩＮ，ＢＡＳＥＷＲＡＰ，ＲＥＧＷＲＡＰパラメータは、以下のチャートに示された値の１つを取る。

パラメータ	説　明
ＲＥＧＣＯＵＮＴ	これは、０、２、４、又は８で、リマッピングを行う１６ビットレジスタの数を同定する。ＲＥＧＣＯＵＮＴ未満のレジスタはリマップされ、ＲＥＧＣＯＵＮＴ以上のレジスタは直接アクセスされる。
ＢＡＳＥＩＮＣ	これは、１６ビットレジスタをいくつ使用して、各ループ繰り返しの最後でベース・ポインタがインクリメントされるかを定義する。好ましい実施の形態では、１、２、又は４の値を取れるが、実際には所望すれば、負の値を含む他の値も取れる。
ＢＡＳＥＷＲＡＰ	これはベース計算のシーリングを決める。ベース・ラッピング・モジュラスは、２、４、８の値を取れる。
ＲＥＧＷＲＡＰ	これは、リマップ計算のシーリングを決める。レジスタ・ラッピング・モジュラスは２、４、８の値を取れる。ＲＥＧＷＲＡＰは、ＲＥＧＣＯＵＮＴと等しく選択することができる。

図６に戻り、リマップ・ブロック５６によって様々なパラメータが使用される例を次に示す（この例では、論理及び物理レジスタ値は、特定バンクに対する相対値である。）

if（Logical Register（論理レジスタ）＜ＲＥＧＣＯＵＮＴ）

　Physical Register(物理レジスタ)＝(Logical Register(論理レジスタ)＋Base（ベース）)ＭＯＤ　ＲＥＧＣＯＵＮＴ

else

　Physical Register(物理レジスタ)＝Logical Register（論理レジスタ）

end if

ループの最後で、ループの次の繰り返しが始まる前に、次のベース・ポインタ更新がベース更新論理５８によって行われる。

Base＝（Base＋ＢＡＳＥＩＮＣ）ＭＯＤ　ＢＡＳＥＷＲＡＰ

リマッピング・ループの最後でレジスタ・リマッピングが打ち切られ、すべてのレジスタは物理レジスタとしてアクセスされる。好ましい実施の形態において、

１つのリマッピングＲＥＰＥＡＴだけがどの時点においてもアクティブである。ループは、ネストされたままであるが、ある特定の時点で１つだけがリマッピング変数を更新してよい。ただし、所望するなら、リマッピング繰り返しはネストできるようにする。

本発明の好ましい実施の形態に基づくリマッピング・メカニズムを使用した結果としてのコード密度に関して達成される効果を示すために、以下、典型的なブロック・フィルタ・アルゴリズムについて説明する。まず、ブロック・フィルタ・アルゴリズムの原則について、図７を参照しながら説明する。図７に示されているように、アキュミュレータ・レジスタＡ０は、多数の乗算操作の結果を累算するように備えられている。この乗算操作というのは、係数ｃ０とデータ項目ｄ０との乗算、係数ｃ１とデータ項目ｄ１との乗算、係数ｃ２とデータ項目ｄ２との乗算などである。レジスタＡ１は、乗算操作の同様のセットの結果を累算していくが、今度は、係数がずれて、ｃ０とｄ１、ｃ１とｄ２、ｃ２とｄ３と組み合わせの乗算になる。同様に、レジスタＡ２は、係数値を更にずらして、ｃ０とｄ２、ｃ１とｄ３、ｃ２とｄ４といった組み合わせの乗算の結果を累算する。このシフト、乗算、累算のプロセスが、繰り返され、その結果がレジスタＡ３に入れられる。

本発明の好ましい実施の形態に基づくレジスタ・リマッピングを使用しないと、ブロック・フィルタ命令を実行するには、次のような命令ループが必要となる。

; start with 4 new data values

（４つの新しいデータ値で始める）

ＺＥＲＯ｛Ａ０－Ａ３｝　　; Zero the accumulators

（アキュミュレータをゼロにする）

ＲＥＰＥＡＴ　Ｚ１　　; Ｚ１＝(number of coeffs/4)

（係数の個数）

; do the next four coefficients, on the first time around:

（第１回目に、次の４つの係数で行う）

; a0 += d0*c0+ d1*c1+ d2*c2+ d3*c3

; a1 += d1*c0+ d2*c1+ d3*c2+ d4*c3

```
; a2 += d2*c0+ d3*c1+ d4*c2+ d5*c3
; a3 += d3*c0+ d4*c1+ d5*c2+ d6*c3
MULA  A0, X0.l^, Y0.l,   A0   ;a0 += d0*c0, and load d4
                                (そしてｄ４をロードする)
MULA  A1, X0.h , Y0.l,   A1   ;a1 += d1*c0
MULA  A2, X1.l , Y0.l,   A2   ;a2 += d2*c0
MULA  A3, X1.h , Y0.l^,  A3   ;a3 += d3*c0, and load c4
                                (そしてｃ４をロードする)
MULA  A0, X0.h^, Y0.h,   A0   ;a0 += d1*c1, and load d5
                                (そしてｄ５をロードする)
MULA  A1, X1.l , Y0.h,   A1   ;a1 += d2*c1
MULA  A2, X1.h , Y0.h,   A2   ;a2 += d3*c1
MULA  A3, X0.l , Y0.h^,  A3   ;a3 += d4*c1, and load c5
                                (そしてｃ５をロードする)
MULA  A0, X1.l^, Y1.l,   A0   ;a0 += d2*c2, and load d6
                                (そしてｄ６をロードする)
MULA  A1, X1.h , Y1.l,   A1   ;a1 += d3*c2
MULA  A2, X0.l , Y1.l,   A2   ;a2 += d4*c2
MULA  A3, X0.h , Y1.l^,  A3   ;a3 += d5*c2, and load c6
                                (そしてｃ６をロードする)
MULA  A0, X1.h^, Y1.h,   A0   ;a0 += d3*c3, and load d7
                                (そしてｄ７をロードする)
MULA  A1, X0.l , Y1.h,   A1   ;a1 += d4*c3
MULA  A2, X0.h , Y1.h,   A2   ;a2 += d5*c3
MULA  A3, X1.l , Y1.h^,  A3   ;a3 += d6*c3, and load c7
                                (そしてｃ７をロードする)
NEXT
```

この例において、データ値はレジスタのＸバンクに入れられ、係数値はレジスタのＹバンクに入れられる。第１ステップとして、４つのアキュミュレータ・レ

ジスタＡ０，Ａ１，Ａ２，Ａ３はゼロにセットされる。アキュミュレータ・レジスタがリセットされると、命令ループが開始され、このループはＲＥＰＥＡＴ命令及びＮＥＸＴ命令によって区切られる。Ｚ１の値は、この命令ループが繰り返される回数を示し、また後で述べる理由により、この回数は、実際には、係数の個数（ｃ０，ｃ１，ｃ２など）を４で割った数に等しい。

命令ループには１６の乗算累算命令（ＭＵＬＡ：multiply accumulate instructions）があり、１回目の繰り返しが終わると、その結果、レジスタＡ０，Ａ１，Ａ２，Ａ３は、ＲＥＰＥＡＴ命令と第１のＭＵＬＡ命令との間で上のコードで示される計算の結果を含む。乗算累算操作がどのように動作するかを示すために、最初の４つのＭＵＬＡ命令を考えることにする。最初の命令によって、Ｘバンク・レジスタ・ゼロの最初のすなわち下の１６ビット内のデータ値と、Ｙバンク・レジスタ・ゼロ内の下の１６ビットとが掛け合わされ、その結果がレジスタＡ０に加えられる。これと同時に、Ｘバンク・レジスタ・ゼロの下の１６ビットが再充填の印になり、レジスタのこの部分に新しいデータ値が再充填できることを示す。このように印がつき、図７から明らかなように、データ項目ｄ０が係数ｃ０で乗算されると（これは最初のＭＵＬＡによって表される）、ｄ０は、ブロック・フィルタ命令の残り部分では不要になり、新しいデータ値で置き換えられる。

次に、第２のＭＵＬＡによって、Ｘバンク・レジスタ・ゼロの第２のすなわち上の１６ビットと、Ｙバンク・レジスタ・ゼロの下の１６ビットとが掛け合わされ（これは、図７における，ｄ１　ｘ　ｃ０を表す）。同様に、第３、第４のＭＵＬＡ命令が、ｄ２　ｘ　ｃ０，及びｄ３　ｘ　ｃ０の乗算を行う。図７から明らかなように、これらの４つの計算が行われると、係数Ｃ０は不要となり、レジスタＹ０．１は、再充填の印がつき、他の係数（ｃ４）で上書きできるようになる。

次の４つのＭＵＬＡ命令は、それぞれ、ｄ１ｘｃ１，ｄ２ｘｃ１，ｄ３ｘｃ１，ｄ４ｘｃ１の計算を表す。ｄ１ｘｃ１の計算が終了すると、ｄ１は不要になるので、レジスタＸ０．ｈは再充填ビットの印がつく。同様に、４つの計算すべてが終了すると、係数ｃ１は不要になるので、レジスタＹ０．ｈは再充填用の印が

つ

く。同様に、次の４つのＭＵＬＡ命令は、ｄ２ｘｃ２，ｄ３ｘｃ２，ｄ４ｘｃ２，ｄ５ｘｃ２の計算に対応し、最後の４つの計算は、ｄ３ｘｃ３，ｄ４ｘｃ３，ｄ５ｘｃ３，ｄ６ｘｃ３の計算に対応する。

上記の実施の形態において、リジスタはリマップできず、各乗算操作は、オペランドで指定される特定レジスタによって明示的に再生されなければならない。１６のＭＵＬＡ命令の実行が終了すると、係数ｃ４からｃ７及びデータ項目ｄ４からｄ１０まで、命令ループを繰り返すことができる。また、ループは、繰り返し１回につき４つの係数値で操作するので、係数値の個数は、４の倍数でなければならず、Ｚ１＝係数／４個の計算が行われる。

本発明の好ましい実施の形態におけるリマッピング・メカニズムを使用することによって、命令ループは飛躍的に減らすことができ、４つの乗算累算命令を含むだけになる。さもなければ１６の乗算累算命令が必要になる。このリマッピング・メカニズムを使用すると、コードは以下のように書くことができる。

```
; start with 4 new data values
```
（４つの新しいデータ値で始める）
```
ZERO {A0－A3}          ; Zero the accumulators
```
（アキュミュレータをゼロにする）
```
REPEAT  Z1, X++ n4 w4 r4, Y++ n4 w4 r4;Z1=(number of coefficients)
```
（係数の個数）
```
; Remapping is applied to the X and Y banks.
```
（リマッピングがＸとＹバンクに適用される。）
```
; Four 16 bit registers in these banks are remapped.
```
（これらのバンク内の４つの１６ビット・レジスタがリマップされる。）
```
; The base pointer for both banks is incremented by one on each
; iteration of the loop.
```
（両方のバンクのベース・ポインタが、ループの繰り返しごとに１つインクリメントされる。）
```
;The base pointer wraps when it reaches the fourth register in the bank.
```
（ベース・ポインタは、バンク内の第４のレジスタに到達するとラップ(wrap)する。）
```
MULA  A0, X0.l^, Y0.l,   A0   ;a0 += d0*c0, and load d4
```
（そしてｄ４をロードする）
```
MULA  A1, X0.h,  Y0.l,   A1   ;a1 += d1*c0
MULA  A2, X1.l,  Y0.l,   A2   ;a2 += d2*c0
MULA  A3, X1.h,  Y0.l^,  A3   ;a3 += d3*c0, and load c4
```
（そしてｃ４をロードする）
```
NEXT                          ;go round loop and advance remapping
```
（ループを１周し、リマップを進める）

先に述べたのと同様に、第１のステップで、４つのアキュミュレータ・レジスタＡ０－Ａ３をゼロにセットする。次に、ＲＥＰＥＡＴオペコードとＮＥＸＴオペコードによって区切られる命令ループに入る。ＲＥＰＥＡＴ命令は、以下のように多数のパラメータを持つ。

Ｘ＋＋：　レジスタのＸバンクに、ＢＡＳＥＩＮＣが‘１’であることを示す

。

ｎ４：　ＲＥＧＣＯＵＮＴが‘４’であり、従って、最初の４つのＸバンクレジスタＸ０．ｌからＸ１．ｈがリマップされることを示す。

ｗ４：　レジスタのＸバンクに、ＢＡＳＥＷＲＡＰが‘４’であることを示す

。

Ｙ＋＋：　レジスタのＹバンクに、ＢＡＳＥＩＮＣが‘１’であることを示す

。

ｎ４：　ＲＥＧＣＯＵＮＴが‘４’であり、従って、最初の４つのＹバンクレジスタＹ０．ｌからＹ１．ｈがリマップされることを示す。

ｗ４：　レジスタのＹバンクに、ＢＡＳＥＷＲＡＰが‘４’であることを示す

。

ｒ４：　レジスタのＹバンクに、ＲＥＧＷＲＡＰが‘４’であることを示す。

尚、Ｚ１の値は、先行技術の例では、係数の個数／４に等しくなるが、ここでは、係数の個数と等しくなる。

命令ループの最初の繰り返しで、ベースポインタの値はゼロであり、リマッピングはない。ただし、次にループが実行される時は、ＸバンクもＹバンクもベース・ポインタの値は‘１’であるから、オペランドは次のようにマップされる。

Ｘ０．ｌはＸ０．ｈになる

Ｘ０．ｈはＸ１．ｌになる

Ｘ１．ｌはＸ１．ｈになる

Ｘ１．ｈはＸ０．ｌになる（ＢＡＳＥＷＲＡＰが‘４’だから）

Ｙ０．ｌはＹ０．ｈになる

Ｙ０．ｈはＹ１．ｌになる

Ｙ１．ｌはＹ１．ｈになる

Ｙ１．ｈはＹ０．１になる（ＢＡＳＥＷＲＡＰが‘４’だから）

従って、２回目の繰り返しでは、本発明のリマッピングを含まない先に述べた例における第５から第８番目のＭＵＬＡ命令によって示される計算を、４つのＭＵＬＡ命令が実際に行うことがわかる。同様に、３回目、４回目のループの繰り

返しでは、先行技術コードの第９から第１２番目、そして第１３から第１６番目のＭＵＬＡ命令によって実行された計算が行われる。

従って、上記コードは、先行技術のコードと全く同様のブロック・フィルタ・アルゴリズムを行うわけだが、ループ本体内のコード密度を４倍に改善している。つまり、先行技術では１６の命令が必要であったのに比較して、４つの命令ですむ。

本発明の好ましい実施の形態に基づくレジスタ・リマッピング技術を使用することによって、以下のような利点が得られる。

１．コード密度を改善する。

２．場合によっては、レジスタを空きとして印をしてからピッコロのリオーダ・バッファによって再充填されるまでのレイテンシー(latency)を隠すこともできる。これは増えるコードサイズを捨ててアンローリングループによって実現される。

３．アクセスされるべきレジスタの数を変化させることができる。ループ繰り返し実行数を変化させることによって、アクセスされるレジスタの数を変化させることができる。

４．アルゴリズム開発を簡単にすることができる。適当なアルゴリズムについて、プログラマはアルゴリズムのｎ番目の段に対する１つのコードを生成して、レジスタ・リマッピングを使用して、その公式をデータのスライディング・セットに適用することができる。

上記レジスタ・リマッピング・メカニズムは、本発明の範囲から離れることなく、ある程度の変形が可能であることが明らかになるであろう。例えば、レジスタ１０のバンクは、プログラマによって命令オペランドに指定される以上の物理レジスタを提供することができる。これらの余分のレジスタは直接的にはアクセスできないが、レジスタ・リマッピング・メカニズムでは、これらのレジスタを使用することができる。例えば、先に出した例を考えてみよう。レジスタのＸバンクに、プログラマの使える３２ビットレジスタが４つあり、従って８つの１６ビットレジスタが論理レジスタ・レファレンスによって指定するこができる。レ

ジスタのＸバンクが、実際には、例えば６つの３２ビットレジスタから成る場合、プログラマにとって直接アクセスできない１６ビットレジスタが余分に４つあることになる。しかしながら、これらの４つのレジスタは、リマッピング・メカニズムによって使用可能となり、データ項目の記憶のための付加的レジスタを提供する。

以下のアセンブラ・シンタクス（文法）を使用することができる。

＞＞は、論理右シフト、又は、シフト・オペランドが負であれば、左シフトを意味する（下の＜１scale＞を参照）。

－＞＞は、算術右シフト、又は、シフト・オペランドが負であれば、左シフトを意味する（下の＜scale＞を参照）。

ＲＯＲは、右回転を意味する。

ＳＡＴ（ａ）は、ａの飽和値を意味する（目的レジスタのサイズによって、１６ビット又は３２ビットで飽和する）。特に、１６ビットで飽和するために、＋０ｘ７ｆｆｆより大きいどんな値も＋０ｘ７ｆｆｆで置き換えられ、－０ｘ８０００より小さいどんな値も－０ｘ８０００で置き換えられる。３２ビット飽和は、同様に、極限値＋０ｘ７ｆｆｆｆｆｆｆと－０ｘ８０００００００がある。目的レジスタが４８ビットである場合も、飽和は３２ビットで行われる。

ソース・オペランド１は、次のフォーマットの１つを取ることができる。

＜ src１＞は、〔Ｒｎ¦Ｒｎ．１¦Ｒｎ．ｈ¦Ｒｎ．ｘ〕〔＾〕の短縮形として使用される。別の言い方をするなら、ソース・スペシファイアの７ビットはすべて有効であり、レジスタは３２ビット値として（希望すれば、交換される）、または符号拡張した１６ビット値として読まれる。アキュミュレータに取っては、下の３２ビットだけが読まれる。“＾”は、レジスタ再充填を指定する。

＜ src__１６＞は、〔Ｒｎ．１¦Ｒｎ．ｈ〕〔＾〕の短縮形として使用される１６ビット値だけが読まれる。

＜ src__３２＞は、〔Ｒｎ¦Ｒｎ．ｘ〕〔＾〕の短縮形として使用される。３

２ビット値だけが読まれ、上半分及び下半分は希望すれば交換できる。

ソース・オペランド２は、次のフォーマットの１つを取ることができる。

＜src２＞は、３つのオプションの短縮形として使用される。

－　〔Rn¦Rn．１¦Rn．h¦Rn．x〕〔^〕の形のソース・レジスタ、プラス最終結果のスケール（＜scale＞）。

－　オプションでシフトされた８ビット定数（＜immed__８＞）、ただし、最終結果のスケールはない。

－　６ビット定数（＜immed__６＞）、プラス、最終結果のスケール（＜scale＞）。

＜src２__maxmin＞は、＜src２＞と同じであるが、ただし、スケールは許可されない。

＜src２__shift＞シフト命令は、＜src２＞の限定的サブセットを提供する詳細は上記を参照。

＜src２__par＞　　＜src２__shift＞用である。

第３のオペランドを指定する命令に対して：

＜acc＞は、４つのアキュミュレータ・レジスタ〔A０¦A１¦A２¦A３〕のいずれかを示す短縮形。４８ビットすべてが読まれる。再充填は指定されない。

目的レジスタは次のフォーマットを持つ：

＜dest＞　これは、〔Rn¦Rn．１¦Rn．h¦．１¦〕〔^〕の短縮形。“．”の拡張はない。

レジスタ全部が書かれる（アキュミュレータの場合は、４８ビット）。レジスタへの書き戻しが必要ない場合は、使用されるレジスタは重要でない。アセンブラが、目的レジスタの省略をサポートし、書き戻しの必要がないこと、又は“．１”つまり、書き戻しは必要ないが結果が１６ビット量であるかのようにフラグをセットすべきであることを示す。^は、値が出力ＦＩＦＯに書き込まれることを示す。

＜scale＞　これは、代数スケールの数を表す。１４のスケールが使用できる。

ＡＳＲ　＃０，１，２，３，４，６，８，１０

ＡＳＲ　＃１２から１６

ＬＳＬ　＃１

＜immed__８＞　これは、符号無し８ビット即値を表す。これは、０、８、１６、又は２４シフトで左回転された１バイトから成る。従って、０ｘＹＺ０００００，０ｘ００ＹＺ００００、０ｘ００００ＹＺ００、０ｘ０００００ＹＺの値が、任意のＹＺに対してエンコードできる。回転は、２ビット量としてエンコードされる。

＜imm__６＞　これは、符号無し６ビット即値を表す。

＜ＰＡＲＡＭＳ＞　これは、レジスタ・リマッピングを指定し、次のフォーマットを持つ：　＜ＢＡＮＫ＞＜ＢＡＳＩＣ＞ｎ＜ＲＥＮＵＭＢＥＲ＞ｗ＜ＢＡＳＥＷＲＡＰ＞

＜BANK＞	〔X¦Y¦Z〕を取れる
＜BASEINC＞	〔＋＋¦＋１¦＋２¦＋４〕を取れる
＜RENUMBER＞	〔０¦２¦４¦８〕を取れる
＜BASEWRAP＞	〔２¦４¦８〕を取れる

＜cond＞という表現は、以下の条件コードの任意の１つの短縮形である。尚、エンコーディングは、ＡＲＭと少し異なる。それは、符号無しＬＳ及びＨＩコードは、より役立つ符号付きオーヴァーフロー／アンダーフローのテストで置き換えられているからである。Ｖフラグ及びＮフラグは、ピッコロ上で、ＡＲＭとは違う方法でセットされるので、条件テストからフラグ・チェックへの翻訳も、ＡＲＭとは異なる。

0000	ＥＱ	Ｚ＝０	最後の結果はゼロであった。
0001	ＮＥ	Ｚ＝１	最後の結果は非ゼロであった。
0010	ＣＳ	Ｃ＝１	shift/ＭＡＸ操作の後で使用される。
0011	ＣＣ	Ｃ＝０	

0100	ＭＩ／ＬＴ	Ｎ＝１	最終結果が負であった。
0101	ＰＬ／ＧＥ	Ｎ＝０	最終結果が正であった。
0110	ＶＳ	Ｖ＝１	最終結果で符号付きオーバーフロー／飽和
0111	ＶＣ	Ｖ＝０	最終結果でオーバーフロー／飽和なし
1000	ＶＰ	Ｖ＝１　＆　Ｎ＝０	最終結果でオーバーフロー正
1001	ＶＮ	Ｖ＝１　＆　Ｎ＝１	最終結果でオーバーフロー負
1010	未使用		
1011	未使用		
1100	ＧＴ	Ｎ＝０　＆　Ｚ＝０	
1101	ＬＥ	Ｎ＝１｜Ｚ＝１	
1110	ＡＬ		
1111	未使用		

ピッコロが扱うのは符号付き量であるから、符号無しＬＳ及びＨＩ条件は、落とされ、オーバーフローの方向を記述するＶＰとＶＮで置き換えられている。ＡＬＵの結果は４８ビット幅であるから、ＭＩとＬＴが、同様にＰＬとＧＥが同じ機能を行う。

すべての操作は、特に注意書のない限り、符号付きである。

第１条件コード及び第２条件コードは、それぞれ、次のものから成る。

Ｎ　負

Ｚ　ゼロ

Ｃ　キャリー／符号無しオーバーフロー

Ｖ　符号付きオーバーフロー

算術命令は、並列命令と「フル幅」命令の２つに分けることができる。「フル幅」命令というのは、一次フラグをセットするだけであるのに対して、並列オペレータは、結果の上１６ビット半分と下１６ビット半分とに基づき、一次フラグと２次フラグをセットする。

Ｎ，Ｚ，Ｖフラグは、スケールを適用した後に、目的に書き込まれる前に、ＡＬＵの結果に基づいて計算される。ＡＳＲは常に、結果を記憶するのに必要なビット数を減らすが、ＡＳＬだと、それを増やす。これを避けるために、ピッコロ

は、ＡＳＬスケールが適用された場合、４８ビットの結果を削って、ゼロ検出及びオーバーフローが行われるビット数を制限する。

Ｎフラグの計算は、符号付き算術計算が行われると推定して、行われる。それは、オーバーフローが起きた場合、結果の最上位ビットはＣフラグかＮフラグであり、それは、入力オペランドが符号付きか符号無しかによるからである。

Ｖフラグは、選択された目的に結果を書き込んだ結果、精度の損失があるか否かを示す。書き戻しが選択されなかった場合も、「サイズ」は含まれており、オーバーフロー・フラグは正しくセットされる。オーバーフローが起きるのは、次の場合である。

－　結果が、－２＾１５から２＾１５－１の範囲にないのに１６ビットレジスタに書き込んだ場合。

－　結果が、－２＾３１から２＾３１－１の範囲にないのに３２ビットレジスタに書き込んだ場合。

並列加算／減算命令は、結果の上半分及び下半分に独立にＮ，Ｚ，Ｖフラグをセットする。

アキュミュレータに書き込みを行うと、３２ビットレジスタに書き込まれたかのように、Ｖフラグがセットされる。

飽和絶対命令（ＳＡＢＳ）も、入力オペランドの絶対値が指定された目的に合わないと、オーバーフロー・フラグをセットする。

キャリー・フラグは、加算と減算命令によりセットされ、ＭＡＸ／ＭＩＮ，ＳＡＢＳ、ＣＬＢ命令によって「バイナリー」フラグとして使用される。乗算操作を含む他のすべての命令は、（単数または複数の）キャリー・フラグを保存する。

加算と減算操作については、キャリーは、ビット３１又はビット１５又は目的が３２ビット幅であるか１６ビット幅であるかの、結果によって生成される。

標準的算術命令は、フラグのセット方法によって、多くのタイプに分類することができる。

加算命令、減算命令の場合、Ｎビットがセットされると、すべてのフラグが保存される。Ｎビットがセットされないと、フラグは、次のように更新される。

Ｚがセットされるのは、フル４８ビット結果が０だった場合。

Ｎがセットされるのは、フル４８ビット結果にビット４７のセットがあった場合（負だった場合）。

Ｖがセットされるのは：

目的レジスタが１６ビットであり、符号付き結果が１６ビットレジスタに合わない（－２＾１５＜＝ｘ＜２＾１５の範囲にない）場合

目的レジスタが３２／４０ビットレジスタであり、符号付き結果が３２ビットに合わない場合

＜dest＞が３２又は４０ビットレジスタである場合でＣフラグがセットされるのは、＜scr１＞と＜scr２＞を合計してビット３１からキャリーがある時、又は、＜scr１＞から＜scr２＞を減算してビット３１から借り（borrow）が生じない時（ＡＲＭ上と同じキャリー）。＜dest＞が１６ビットレジスタである場合でＣフラグがセットされるのは、合計のビット１５からキャリーがある時。

２次フラグ（ＳＺ，ＳＮ，ＳＶ，ＳＣ）は保存される。

４８ビットレジスタから乗算又は累算を行う命令の場合。

Ｚがセットされるのは、フル４８ビット結果が０だった場合。

Ｎがセットされるのは、フル４８ビット結果にビット４７のセットがあった場合（負だった場合）。

Ｖがセットされるのは：（１）目的レジスタが１６ビットであり、符号付き結果が１６ビットレジスタに合わない（－２＾１５＜＝ｘ＜２＾１５の範囲にない）場合、（２）目的レジスタが３２／４８ビットレジスタであり、符号付き結果が３２ビットに合わない場合

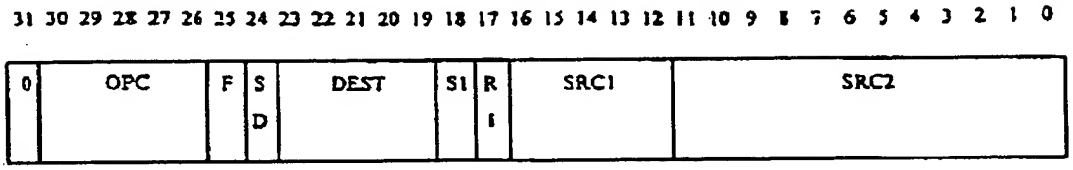
Ｃは保存される。

２次フラグ（ＳＺ，ＳＮ，ＳＶ，ＳＣ）は保存される。

論理操作、並列加算及び減算、max及びmin、シフトなどを含むその他の命令は、以下のようにカバーされる。

加算命令、減算命令は、２つのレジスタを加算又は減算し、結果をスケールして、レジスタに戻して記憶させる。オペランドは、符号付き値として扱われる。

不飽和変種に対するフラグ更新は、オプショナルであり、Ｎを命令の最後に付け足すことによって抑制することもできる。



ＯＰＣは、命令のタイプを指定する。

100N0	dest=(src1 + src2) (－＞＞ scale) (,N)
110N0	dest=(src1 - src2) (－＞＞ scale) (,N)
10001	dest=SAT((src1 + src2) (－＞＞ scale))
11001	dest=SAT((src1 - src2) (－＞＞ scale))
01110	dest=(src2 - src1) (－＞＞ scale)
01111	dest=SAT((src2 - src1) (－＞＞ scale))
101N0	dest=(src1 + src2 + Carry) (－＞＞ scale) (,N)
111N0	dest=(src1 - src2 + Carry-1) (－＞＞ scale) (,N)

ニューモニックス：

100N0	ＡＤＤ {N}	<dest>, <src1>, <src2> {,<scale>}
110N0	ＳＵＢ {N}	<dest>, <src1>, <src2> {,<scale>}
10001	ＳＡＤＤ	<dest>, <src1>, <src2> {,<scale>}
11001	ＳＳＵＢ	<dest>, <src1>, <src2> {,<scale>}
01110	ＲＳＢ	<dest>, <src1>, <src2> {,<scale>}
01111	ＳＲＳＢ	<dest>, <src1>, <src2> {,<scale>}
101N0	ＡＤＣ {N}	<dest>, <src1>, <src2> {,<scale>}
111N0	ＳＢＣ {N}	<dest>, <src1>, <src2> {,<scale>}

アセンブラは以下のオペコードをサポートする

CMP　<src1>, <src2>

CMN　<src1>, <src2>

ＣＭＰは、レジスタ書き込みディスエーブル（disabled）のフラグをセットす

る減算であり、ＣＭＮは、レジスタ書き込みディスエーブルのフラグをセットする加算である。

フラグ：

これについては、上記の通り。

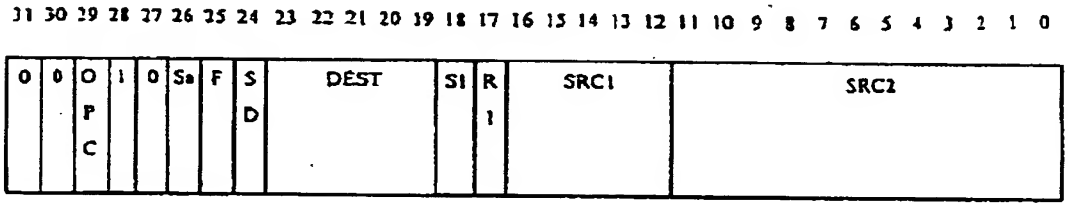
含める理由

ＡＤＣは、shift／ＭＡＸ／ＭＩＮ操作に続いてレジスタの下にキャリーを挿入するのに使える。また、３２／３２割算を行うのにも使用される。さらに、拡張精密加算を提供する。Ｎビットを加算することによって、フラグを細かく制御することができ、特にキャリーを制御できる。これにより、１ビットにつき２サイクルで、３２／３２ビット割算ができる。

飽和加算及び減算が、Ｇ．７２９などに必要である。

カウンタのインクリメント／ディクリメント。ＲＳＢは計算シフト（ｘ＝３２－ｘが普通の操作）に使える。飽和ＲＳＢは、飽和否定（Ｇ．７２９で使用される）に必要である。

加算／減算累算命令は、累算及びスケーリング／飽和を伴う加算及び減算を行う。乗算累算命令と違って、アキュミュレータ番号は、目的レジスタと独立に指定することはできない。目的レジスタの下２ビットは、累算に使う４８ビットアキュミュレータの番号、accを与える。従って、ＡＤＤＡ　Ｘ０，Ｘ１，Ｘ２，Ａ０及びＡＤＤＡ　Ａ３，Ｘ１，Ｘ２，Ａ３は有効であるが、ＡＤＤＡ　Ｘ１，Ｘ１，Ｘ２，Ａ０は無効である。このクラスの命令では、結果はレジスタに書き戻されなければならず、目的領域の書き戻し無しエンコーディングは許可されない。



ＯＰＣは、命令のタイプを指定する。以下において、accは（ＤＥＳＴ〔１：０〕）である。Ｓａビットは、飽和を示す。

動作（ＯＰＣ）：

0　dest = {ＳＡＴ} (acc + (src1 + src2))　{－＞＞ scale}

1　dest = {ＳＡＴ} (acc + (src1 - src2))　{－＞＞ scale}

ニューモニック：

0　{S} ADDA　<dest>, <src1>, <src2>, <acc > {,<scale>}

1　{S} SUBA　<dest>, <src1>, <src2>, <acc > {,<scale>}

コマンドの前のＳは飽和を示す。

フラグ：

上記を参照

含める理由

ＡＤＤＡ（加算累算）命令は、１サイクルにつき、整数アレーの２ワードとアキュミュレータ（例えば、それらの平均を見つけるのに）の和を取るのに使える。ＳＵＢＡ（減算累算）命令は、差の和を計算するのに（例えば相関のために）使え、２つの別個の値を減算して、その差を第３のレジスタに加える。

＜acc＞とは異なる＜dest＞を使用することによって、丸め（rounding）をともなう加算をすることもできる。例えば、Ｘ０＝（Ｘ１＋Ｘ２＋１６３８４）＞＞１５は、１６３８４をＡ０に保持しながら１サイクルで行うことができる。丸め付定数加算は、ＡＤＤＡ　Ｘ０，Ｘ１，＃１６３８４，Ａ０で行うことができる。

ビットの正確な導入には：

sum of ((a__i*b__i)＞＞k)　（一般的にはTrueSpeechで使用される）

標準ピッコロ・コードは以下のようになる：

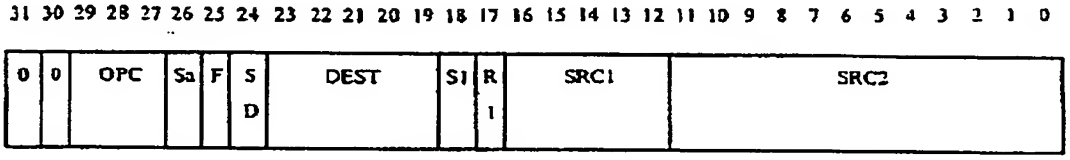
```
ＭＵＬ    t1,   a__0, b__0, ASR#k
ＡＤＤ    ans, ans, t1
ＭＵＬ    t2, a__1, b__1, ASR#k
ＡＤＤ    ans, ans, t2
```

このコードには２つの問題がある。１つは長すぎること、もう１つは、加算が４８ビット精密加算ではなくガードビットが使用できないこと。これに対処するには、ＡＤＤＡを使うことである。

```
ＭＵＬ    t1, a__0, b__0, ASR#k
ＭＵＬ    t2, a__1, b__1, ASR#k
ＡＤＤＡ  ans, t1, t2, ans
```

これにより、２５％のスピードアップが得られる、４８ビット精度が保持される。

並列命令における加算／減算は、３２ビットレジスタに対（ペア）で保持される２つの符号付き１６ビット量で行われる。一次条件コードフラグは、最上位１６ビットの結果からセットされ、二次フラグは、下位半分から更新される。これらの命令のソースとして指定できるのは３２ビットレジスタだけであるが、値は、ハーフワード交換できる。各レジスタの個々の半分は、符号付き値として扱われる。計算及びスケーリングは、精度損失無しで行われる。従って、ＡＤＤＡＤＤ　Ｘ０，Ｘ１，Ｘ２，ＡＳＲ＃１は、Ｘ０の上半分及び下半分における正しい平均を生成する。各命令にはオプショナル飽和が提供され、それには、Ｓａビットをセットする。



ＯＰＣが操作を定義する。

動作（ＯＰＣ）：

```
000   dest.h=(src1.h + src2.h)－＞＞ {scale } ,
      dest.1=(src1.1 + src2.1)－＞＞ {scale }
001   dest.h=(src1.h + src2.h)－＞＞ {scale } ,
      dest.1=(src1.1 - src2.1)－＞＞ {scale }
100   dest.h=(src1.h - src2.h)－＞＞ {scale } ,
      dest.1=(src1.1 + src2.1)－＞＞ {scale }
101   dest.h=(src1.h - src2.h)－＞＞ {scale } ,
      dest.1=(src1.1 - src2.1)－＞＞ {scale }
```

Ｓａビットがセットされている場合、各和／差分は独立に飽和する。

ニューモニック：

000　{Ｓ} ADDADD　<dest>, <src1_32>, <src2_32> {,<scale>}

001　{Ｓ} ADDSUB　<dest>, <src1_32>, <src2_32> {,<scale>}

100　{Ｓ} SUBADD　<dest>, <src1_32>, <src2_32> {,<scale>}

101　{Ｓ} SUBSUB　<dest>, <src1_32>, <src2_32> {,<scale>}

コマンドの前のＳは飽和を示す。

アセンブラは以下のものもサポートする

CMNCMN　　<dest>, <src1_32>, <src2_32> {,<scale>}

CMNCMP　　<dest>, <src1_32>, <src2_32> {,<scale>}

CMPCMN　　<dest>, <src1_32>, <src2_32> {,<scale>}

CMPCMP　　<dest>, <src1_32>, <src2_32> {,<scale>}

書き戻しなしの標準命令によって生成される。

フラグ

Ｃがセットされるのは、２つの上の１６ビット半分を加算する時に、ビット１５のキャリーがある場合。

Ｚがセットされるのは、上の１６ビット半分の和が０である場合。

Ｎがセットされるのは、上の１６ビット半分の和が負である場合。

Ｖがセットされるのは、上の１６ビット半分の符号付き１７ビット和が１６ビットに当てはまらない（ポスト・スケール）場合。

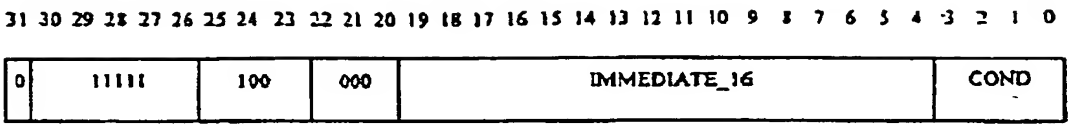
ＳＺ，ＳＮ，ＳＶ，ＳＣが、同様に、下の１６ビット半分に対してセットされる。

含める理由

並列加算及び減算命令は、単一３２ビットレジスタに保持される複素数を操作するのに使用でき、ＦＦＴカーネルで使用される。また、１６ビットデータのベクトルの単純な加算／減算にも使え、１サイクルで２つの要素を処理することができる。

ブランチ（条件付き）命令は、制御フローにおける条件付き変更を行うことを

可能とする。ピッコロは、取られたブランチを実行するのに３サイクル使う。



動作

一次フラグに基づき＜cond＞が保持されれば、オフセットによるブランチ。

オフセットは、符号付き１６ビット番号のワードである。この時、オフセットの範囲は、－３２７６８から＋３２７６７ワードに制限される。

アドレス計算は次のようにされる。

目的アドレス＝ブランチ命令アドレス＋４＋オフセット

ニューモニック：

Ｂ＜cond＞＜destination_label＞

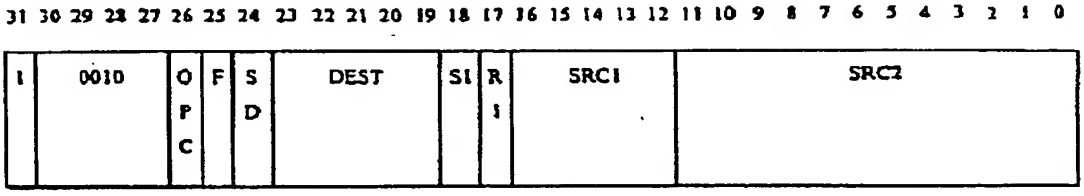
フラグ：

影響されない

含める理由：

殆どのルーチンで非常に役立つ。

条件付き加算又は減算命令は、条件付きでsrc２をsrc１へ加算または減算する。



ＯＰＣが命令のタイプを指定する。

動作（ＯＰＣ）：

0　if(carry set)temp=src1-src2 else temp=src1+src2

dest = temp {－＞＞ scale}

1　if(carry set)temp=src1-src2 else temp=src1+src2

dest = temp {－＞＞ scale} BUT if scale is a shift left

(もしスケールが左シフトなら)

then the new value of carry(from src1-src2 or src1+src2)is shifted into the bottom.

(src1-src2 又は src1+src2からの) キャリーの新しい値がボトムにシフトされる)

ニューモニック：

0　CAS　　<dest>, <src1>, <src2> {,<scale>}

1　CASC　　<dest>, <src1>, <src2> {,<scale>}

フラグ：

上記参照

含める理由

条件付き加算または減算命令により、効率のよい除算コードを構成することができる。

例１：　Ｘ０にある３２ビット符号無し値を、Ｘ１にある１６ビット符号無し値で割る（Ｘ０＜（Ｘ１＜＜１６）且つＸ１．ｈ＝０と仮定する）。

```
LSL  X1, X1, #15        ;除数をシフトアップする
SUB  X1, X1, #0         ;キャリーフラグをセットする
REPEAT  #16
CASC  X0, X0, X1, LSL#1
NEXT
```

ループの最後で、Ｘ０．１は除算の商を保持する。余りは、キャリーの値に従って、Ｘ０．ｈから復元される。

例２：　ＸＯにある３２ビット正の値を、Ｘ１にある３２ビット正の値で割り、早く終了する。

```
MOV       X2, #0          ; 商をクリアする
LOG       Z0, X0          ; ビット数Ｘ０分シフトできる
LOG       Z1, X1          ; ビット数Ｘ１分シフトできる
SUBS      Z0, Z1, Z0      ; Ｘ１がシフトして１がマッチする
BLT       div __end       ; Ｘ１＞Ｘ０なので答は０
LSL       X1, X1, Z0      ; マッチした場合
ADD       Z0, Z0, #1      ; 行うべきテストの回数
SUBS      Z0, Z0, #0      ; キャリーをセットする
REPEAT    Z0
CAS       X0, X0, X1, LSL#1
ADCN      X2, X2, X2
```

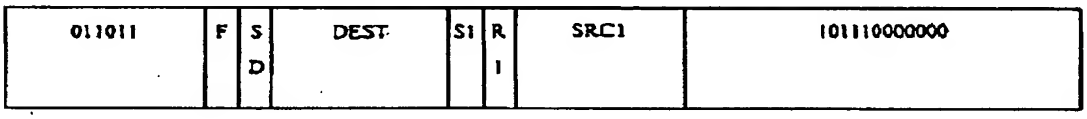
NEXT

div__end

最後に、Ｘ２が商を保持し、余りは、Ｘ０から復元される。

カウント・リーディング・ビット命令により、データが正規化される。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



動作

destは、src１にある値が左にシフトされるべき場所数にセットされて、ビット３１がビット３０と異なるようにする。これは０－３０の範囲の値であるが、例外として、src１が－１又は０の場合は、３１が戻される。

ニューモニック

ＣＬＢ　＜dest＞，＜src1＞

フラグ

Ｚがセットされるのは、結果が０の時。

Ｎはクリアされる。

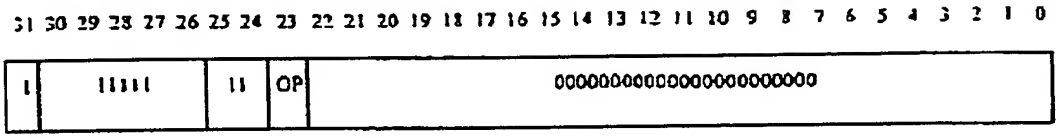
Ｃがセットされるのは、src１が－１又は０の時。

Ｖは未使用。

含む理由：

正規化に必要なステップ

ピッコロの実行を止めるには、Halt及びBreakpoint命令がある。



ＯＰＣは命令のタイプを指定する。

動作（ＯＰＣ）

０　ピッコロの実行が止められ、Haltビットがピッコロ状態レジスタにセットされる。

１　ピッコロの実行が止められ、Breakビットがピッコロ状態レジスタにセットされ、ＡＲＭが中断され、ブレークポイントに到達したことを知らせる。

ニューモニック

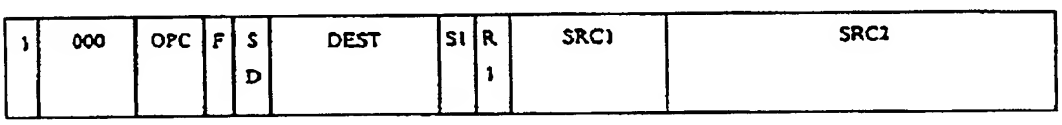
０　ＨＡＬＴ

１　ＢＲＥＡＫ

フラグ

影響されない。

論理演算命令は、３２又は１６ビットレジスタ上で論理演算を行う。オペランドは、符号無し値として扱われる。



ＯＰＣは、実行すべき論理操作をエンコードする。

動作（ＯＰＣ）：

00　dest = (src1 & src2) {－＞＞ scale}

01　dest = (src1¦src2) {－＞＞ scale}

10　dest = (src1 &～src2) {－＞＞ scale}

11　dest = (src1＾src2) {－＞＞ scale}

ニューモニック：

00　ＡＮＤ　<dest>, <src1>, <src2>, {,<scale>}

01　ＯＲＲ　<dest>, <src1>, <src2>, {,<scale>}

10　ＢＩＣ　<dest>, <src1>, <src2>, {,<scale>}

11　ＥＯＲ　<dest>, <src1>, <src2>, {,<scale>}

アセンブラが以下のオペコードをサポートする

ＴＳＴ　<src1>, <src2>

ＴＥＱ　<src1>, <src2>

ＴＳＴは、レジスタ書き込みがディスエーブルされたＡＮＤである。ＴＥＱはレジスタ書き込みがディスエーブルされたＥＯＲである。

フラグ

Ｚがセットされるのは、結果が全て０の時。

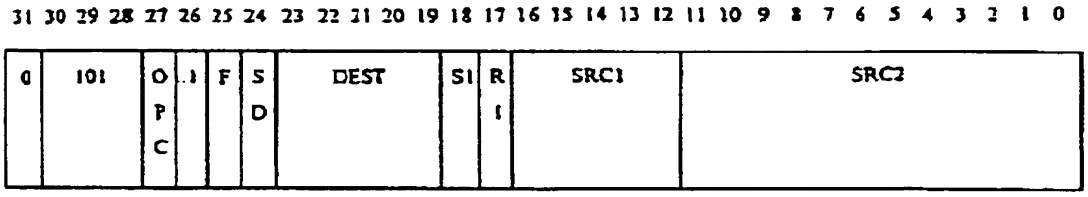
Ｎ，Ｃ，Ｖは保存される。

ＳＺ，ＳＮ，ＳＣ，ＳＶは保存される。

含む理由：

スピーチ圧縮アルゴリズムは、情報をエンコードするために、パックされたビット領域を使用する。ビットマスク命令は、これらの領域の抽出／パック化を助ける。

Ｍａｘ及びＭｉｎ操作命令は、最大及び最小操作を実行する。



ＯＰＣは命令のタイプを指定する。

動作（ＯＰＣ）：

0　　dest = (src1<＝src2) ？src1：src2

1　　dest = (src1>src2) ？src1：src2

ニューモニック：

0　ＭＩＮ　<dest>, <src1>, <src2>

1　ＭＡＸ　<dest>, <src1>, <src2>

フラグ

Ｚがセットされるのは、結果が０の時。

Ｎがセットされるのは、結果が負の時。

Ｃ　Maxでは、ｓｒｃ２＞＝ｓｒｃ１（ｄｅｓｔ＝ｓｒｃ１の場合）の時にセットされる。Minでは、ｓｒｃ２＞＝ｓｒｃ１（ｄｅｓｔ＝ｓｒｃ２の場合）の時にセットされる。

Ｖ保存される

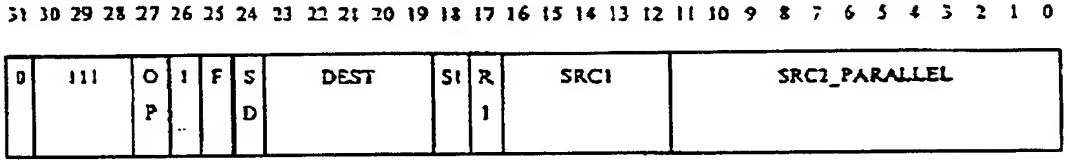
含む理由：

信号の強さを見るために、多数のアルゴリズムがサンプルをスキャンして、サンプルの絶対値の最大／最小を決める。これに、ＭＡＸ，ＭＩＮ操作が使用できる。信号の最初の最大値か最後の最大値のどちらを見つけたいかによって、オペランドｓｒｃ１及びｓｒｃ２は、交換することができる。

ＭＡＸ　Ｘ０，Ｘ０，＃０は、Ｘ０を正の数に変換し下をクリップする。

ＭＩＮ　Ｘ０，Ｘ０，＃２５５は、Ｘ０の上をクリップする。これは、グラフィック処理に役立つ。

並列命令におけるＭＡＸ，ＭＩＮ操作は、並列１６ビットデータ上で最大値、最小値操作を行う。



ＯＰＣは、命令のタイプを指定する。

動作（ＯＰＣ）：

0　dest.l＝ (src1.l＜＝src2.l) ？src1.l：src2.l

dest.h＝ (src1.h＜＝src2.h) ？src1.h：src2.h

1　dest.l＝ (src1.l＞src2.l) ？src1.l：src2.l

dest.h＝ (src1.h＞src2.h) ？src1.h：src2.h

ニューモニック：

0　ＭＩＮＭＩＮ　＜dest＞, ＜src1＞, ＜src2＞

1　ＭＡＸＭＡＸ　＜dest＞, ＜src1＞, ＜src2＞

フラグ

Ｚがセットされるのは、結果の上１６ビットがゼロの場合。

Ｎがセットされるのは、結果の上１６ビットが負の場合。

Ｃ　Max：　src２．ｈ＞＝scrc１．ｈ（dest＝src１の場合）の時にセットされる。

Min：　src２．ｈ＞＝scrc１．ｈ（dest＝src２の場合）の時にセットされる。

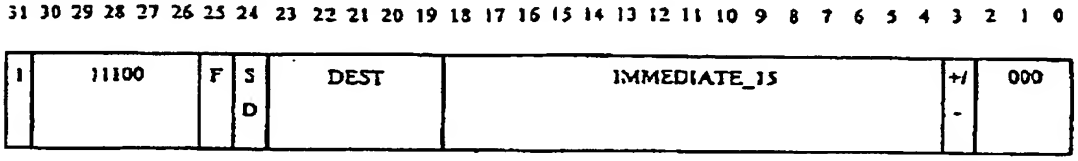
Ｖ保存される

ＳＺ，ＳＮ，ＳＣ，ＳＶは、同様に、下１６ビット半分用にセットされる。

含む理由：

３２ビットＭａｘ，Ｍｉｎについて。

Move Long Immeidate Operation命令により、レジスタは、どの符号付き１６ビットの符号拡張値をセットされることができる。これらの命令のうち２つは、３２ビットレジスタに任意の値にセットすることができる（連続する高位半分と低位半分にアクセスすることによって）。レジスタ間の移動については、選択操作を参照。



ＭＯＶ＜dest＞，＃＜imm__16＞

アセンブラは、ＭＯＶ命令を使用して非インターロックＮＯＰ操作を提供することができる。つまり、ＮＯＰは、ＭＯＶ，＃０と等価である。

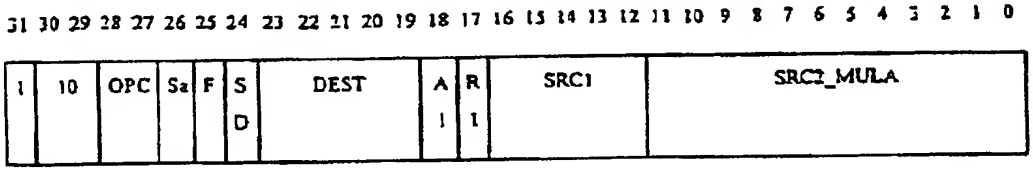
フラグ

フラグは影響されない。

含む理由：

レジスタ／カウンタをイニシアライズする。

乗算累算操作命令は、符号付き乗算を行い、累算または退出(deaccumulation)、スケーリング及び飽和を伴う。



ＯＰＣ領域は命令のタイプを特定する。

動作（ＯＰＣ）：

00　　dest = (acc +(src1*src2)) {−>> scale}

01　　dest = (acc -(src1*src2)) {−>> scale}

各場合、Ｓａビットがセットされていれば、結果は目的に書き込まれる前に飽和される。

ニューモニック：

00　{S} MULA <dest>, <src1 __16>,<src2 __16>,<acc> {,<scale>}

01　{S} MULS <dest>, <src1 __16>,<src2 __16>,<acc> {,<scale>}

コマンドの手前のＳは飽和を示す。

フラグ：

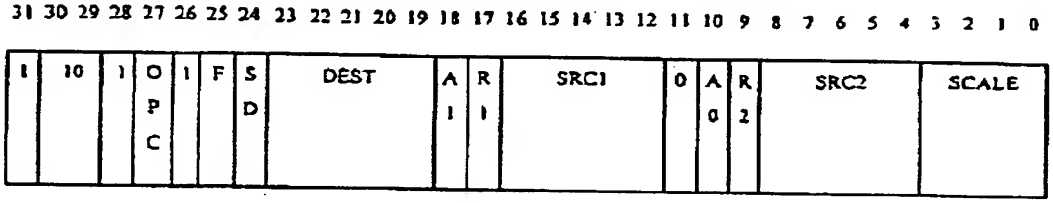
上記を参照。

含む理由：

１サイクル保持されたＭＵＬＡがＦＩＲコードに必要である。ＭＵＬＳは、ＦＦＴバタフライで使用される。また、ＭＵＬＡは、丸め（rounding）付き乗算に役立つ。例えば、Ａ０＝（Ｘ０＊Ｘ１＋１６３８４）＞＞１５は、１６３８４を別のアキュミュレータ（例えばＡ１）に保持することによって、１つのサイクル

で行うことができる。ＦＦＴカーネルには異なった＜dest＞及び＜acc＞が必要である。

Multiply Double Operation命令は、符号付き乗算を行い、結果をダブルにしてから累算又は退出、スケーリング、飽和を行う。



ＯＰＣは命令のタイプを指定する。

動作（ＯＰＣ）

0　　dest=SAT ((acc + SAT (2*src1*src2)) {−>> scale}

1　　dest=SAT ((acc - SAT (2* src1* src2)) {−>> scale}

ニューモニック：

0　SMLDA　<dest>,<src1__16>, <src2__16>,<acc>　{,<scale>}

1　SMLDS　<dest>,<src1__16>, <src2__16>,<acc>　{,<scale>}

フラグ：

上記参照

含む理由：

ＭＬＤ命令は、Ｇ．７２９など、分数（fractional）算術を使用するアルゴリズムにとって必要である。殆どのＤＳＰは、累算又は書き戻しの前に乗数の出力において１ビット左にシフトさせることのできる分数モードを提供する。これを特定命令としてサポートすることにより、プログラマにはより大きなフレキシビリティーが与えられる。Ｇシリーズの基本操作のいくつかと同等の名前を以下に示す。

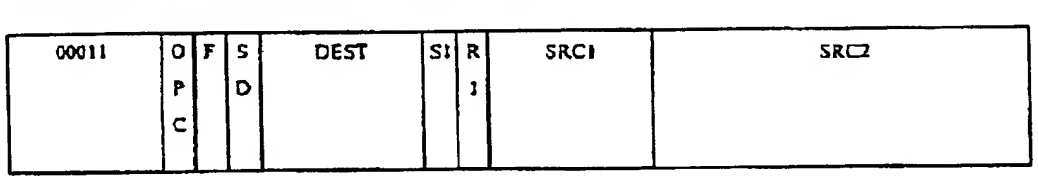
Ｌ__msu＝＞ＳＭＬＤＳ

Ｌ__mac＝＞ＳＭＬＤＡ

これらは、１ビット左シフトする時に乗数の飽和を利用する。一連の分数の乗算・累算が必要な場合、精度のロスなしに、ＭＵＬＡを使うことができ、その和は、３３．１４フォーマットで保持される。必要なら、左シフト及び飽和を最後

に利用して、１．１５フォーマットに変換することができる。

乗算演算命令は、符号付き乗算、及びオプショナルなスケーリング／飽和を行う。ソース・レジスタ（１６ビットのみ）は、符号付き数として扱われる。



ＯＰＣは命令のタイプを指定する。

動作（ＯＰＣ）：

0　dest= (src1*src2) {−>> scale}

1　dest=SAT (src1*src2) {−>> scale}

ニューモニック：

0　MUL　<dest>, <src1_16>, <src2> {,<scale>}

1　SMUL　<dest>, <src1_16>, <src2> {,<scale>}

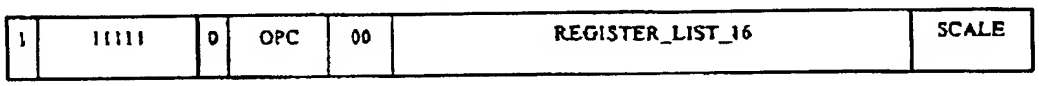
フラグ

上記を参照。

含む理由。

符号付き且つ飽和乗算は、多くの処理で必要となる。

Register List操作は、複数のレジスタのセット（集合）に操作を行う時に使用される。Empty and Zero命令は、ルーチンを始める前に、あるいはルーチンとルーチンとの間で、レジスタの選択をリセットするのに使用する。Output命令を使って、レジスタのリストの内容を出力ＦＩＦＯに記憶することができる。



ＯＰＣは命令のタイプを指定する。

動作（ＯＰＣ）：

000　(k=0; k＜16; k++)用

if bit k of the register list is set then register k is marked as being empty.（レジスタリストのビットｋがセットされると、レジスタｋは空きの印になる。）

001　(k=0; k＜16; k++)用

if bit k of the register list is set then register k is set to contain 0.（レジスターリストのビットｋがセットされると、レジスタｋが０を含むようセットされる。）

010　未定義

011　未定義

100　(k=0; k＜16; k++)用

if bit k of the register list is set then （レジスタリストのビットｋがセットされると、）(register k－＞＞scale)is written to the output FIFO.（が出力ＦＩＦＯに書き込まれる。）

101　(k=0; k＜16; k++)用

if bit k of the register list is set then （レジスタリストのビットｋがセットされると、）(register k－＞＞scale)is written to the output FIFO and register k is marked as being empty.（が出力ＦＩＦＯに書き込まれ、レジスタｋが空きの印になる。

110　(k=0; k＜16; k++)用

if bit k of the register list is set then （レジスタリストのビットｋがセットされると、）ＳＡＴ (register k－＞＞scale)is written to the output FIFO.（が出力ＦＩＦＯに書き込まれる。）

111　(k=0; k＜16; k++)用

if bit k of the register list is set then （レジスタリストのビットkがセットされると、）ＳＡＴ (register k－＞＞scale)is written to the output FIFO and register k is marked as being empty.（が出力ＦＩＦＯに書き込まれ、レジスタｋが空きの印になる。）

ニューモニック：

000	EMPTY	<register_list>
001	ZERO	<register_list>
010	Unused	
011	Unused	
100	OUTPUT	<register_list> {,<scale>}
101	OUTPUT	<register_list>^ {,<scale>}
110	SOUTPUT	<register_list> {,<scale>}
111	SOUTPUT	<register_list>^ {,<scale>}

フラグ

影響されない

例

ＥＭＰＴＹ　　｛Ａ０，Ａ１，Ｘ０－Ｘ３｝

ＺＥＲＯ　　｛Ｙ０－Ｙ３｝

ＯＵＴＰＵＴ　｛Ｘ０－Ｙ１｝^

また、アセンブラはシンタクス（文法）をサポートする。

ＯＵＴＰＵＴ　Ｒｎ

その場合、ＭＯＶ^，Ｒｎ命令を使ってレジスタを１つ出力することになる。

ＥＭＰＴＹ命令は、空であるすべてのレジスタが有効データを含む（すなわち、空きでない）まで、止まっている。

リマッピングＲＥＰＥＡＴループ内では、レジスタ・リスト操作は使用されるべきでない。

ＯＵＴＰＵＴ命令が出力用に指定することができるレジスタは８つまでである。

含む理由：

１つのルーチンが終了した後、次のルーチンは、ＡＲＭからデータを受け取れるようすべてのレジスタが空きであることを期待する。これを逐行するために、ＥＭＰＴＹ命令が必要となる。ＦＩＲそのたのフィルタを実行する前に、すべてのアキュミュレータ及び部分的結果がゼロにされなければならない。これには、

ＺＥＲＯ命令が助けとなる。これらの命令は、一連の単一レジスタ移動を置き換えることによってコード密度を改善するよう設計されている。ＯＵＴＰＵＴ命令は、一連のＭＯＶ，Ｒｎ命令を置き換えることによってコード密度を改善するべく含まれる。

リマッピング・パラメータ・移動命令ＲＭＯＶが提供されるので、ユーザ定義のレジスタ・リマッピング・パラメータの構成を取ることができる。

命令エンコーディングは以下の通り。

31	30 29 28 27 26	25 24 23	22 21	20 19 18 17 16 15 14	13 12 11 10 9 8 7	6 5 4 3 2 1 0
1	11111	101	00	ZPARAMS	YPARAMS	XPARAMS

各ＰＡＲＡＭＳ領域は次のエントリから成る：

6	5	4	3	2	1	0
BASEWRAP		BASEINC		0	RENUMBER	

これらのエントリの意味を以下に示す。

パラメータ	説　　明
ＲＥＮＵＭＢＥＲ	リマッピングを行う１６ビットレジスタの数であって、値０，２，４，８を取ることができる。 ＲＥＮＵＭＢＥＲ未満のレジスタはリマップされ、それ以上のレジスタは直接アクセスされる。
ＢＡＳＥＩＮＣ	各ループの最後でベースポインタがインクリメントされる量。
ＢＡＳＥＷＲＡＰ	ベースラッピング・モジュラスが取れる値は2，4，８。

ニューモニック：

ＲＭＯＶ＜ＰＡＲＡＭＳ＞，［＜ＰＡＲＡＭＳ＞］

＜ＰＡＲＡＭＳ＞領域は次のフォーマットを取る。

＜ＰＡＲＡＭＳ＞::＝＜BANK＞＜BASEINC＞ n ＜RENUMBER＞ w ＜BASEWRAP＞

<BANK>::= [X¦Y¦Z]

<BASEINC>::= [++¦＋１¦＋２¦＋４]

<RENUMBER>::= [0¦2¦4¦8]

<BASEWRAP>::= [2¦4¦8]

ＲＭＯＶ命令の使用がリマッピングのアクティブ中だと、その挙動は、ＵＮＰＲＥＤＩＣＡＴＡＢＬＥ（予想不可）である。

フラグ

影響されない。

Repeat命令は、４つのゼロ・サイクル・ループをハードウェアで提供する。ＲＥＰＥＡＴ命令は、新しいハードウェア・ループを定義する。ピッコロは、最初のＲＥＰＥＡＴ命令にハードウェア・ループ０を使用し、最初のrepeat命令に埋め込まれた（nested）ＲＥＰＥＡＴ命令にハードウェア・ループ１を使用し、以下同様である。ＲＥＰＥＡＴ命令は、どのループが使用されているかを指定する必要はない。ＲＥＰＥＡＴ命令は厳密に埋め込まれなければならない。深さ４を越える埋め込みを試みると、挙動は、予想不可となる。

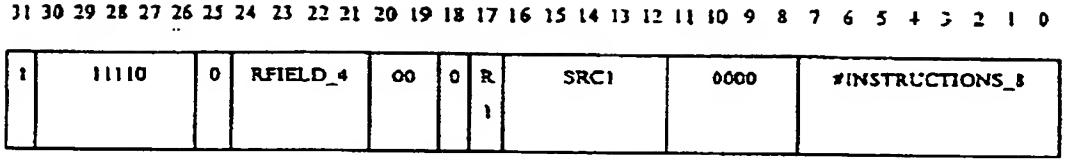
各ＲＥＰＥＡＴ命令は、（ＲＥＰＥＡＴ命令の直後の）ループ内の命令の数を指定し、そのループを何回巡るかの回数（定数またはピッコロレジスタから読み込まれる）を指定する。

ループ内の命令の数が小さい（１又は２）場合、ピッコロはそのループをセットアップするために余分のサイクルを使っても良い。

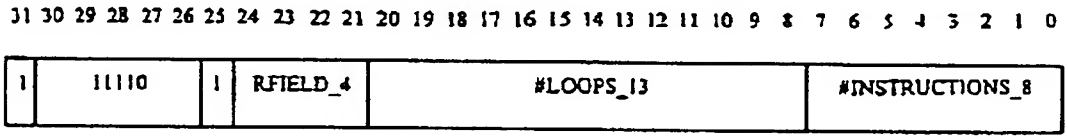
ループ・カウントがレジスタ指定であれば、３２ビットアクセスという意味になる（Ｓ１＝１）が、下の１６ビットだけが意味を持ち、その数は符号無しであるとされる。ループ・カウントがゼロの場合、ループの動作は未定義である。ループ・カウントのコピーが取られ、レジスタはループに影響せずに直接再利用（又は、再充填さえ）できる。

ＲＥＰＥＡＴ命令は、ループ内でレジスタ・オペランドが指定される方法を変えるメカニズムを提供する。詳細は上記の通り。

ループ数がレジスタ指定されたＲＥＰＥＡＴのエンコーディング：



固定されたループ数のＲＥＰＥＡＴのエンコーディング：



ＲＦＩＥＬＤオペランドは、ループ内でどの１６リマッピングパラメータ構成を使用すべきかを指定する。

RFIELD	BASEINC	RENUMBER {X,Y,Z}	BASEWRAP	ニューモニック
0	-	000	-	リマッピングなし
1	-	---	-	ユーザ定義パラメータ
2	1	880	0	X++ n8, Y++ n8
3	1	800	0	X++ n8
4	1	440	0	X++ n4, Y++ n4
5	1	400	0	X++ n4
6	1	220	0	X++ n2, Y++ n2
7	1	200	0	X++ n2
8	2	880	0	X+2 n8, Y+2 n8
9	2	800	0	X+2 n8
10	2	440	0	X+2 n4, Y+2 n4
11	2	400	0	X+2 n4
12	1	820	0	X++ n8, Y++ n2
13	1	444	0	X++ n4, Y++ n4, Z++ n4
14	-	---	-	予約済
15	-	---	-	予約済

アセンブラは、ハードウェア・ループを定義するためにＲＥＰＥＡＴとＮＥＸＴという２つのオペコードを提供する。ＲＥＰＥＡＴはループの始めに行き、ＮＥＸＴはループの最後を区切ることによって、アセンブラはループ本体内にある命令の数を数えることができる。ＲＥＰＥＡＴにとって必要なことは、ループの

数を定数あるいはレジスタとして指定すればよいだけである。例えば：

```
REPEAT    X0
MULA      A0, Y0. l, Z0. l, A0
MULA      A0, Y0. h^, Z0. h^, A0
NEXT
```

これは、２つのＭＵＬＡ命令をＸ０回実行する。また、

```
REPEAT    #10
MULA      A0, X0^, Y0^, A0
NEXT
```

は、１０回乗算累算を行う。

アセンブラは、次のシンタクス（文法）をサポートする。

ＲＥＰＥＡＴ #iterations［, ＜ＰＡＲＡＭＳ＞］

ＲＥＰＥＡＴのために使用するリマッピング・パラメータを指定する。必要なリマッピング・パラメータが前もって定義されたパラメータのセットの１つと等しい場合は、適当なＲＥＰＥＡＴエンコーディングが使用される。そうでなければ、アセンブラはＲＭＯＶを生成してＲＥＰＥＡＴ命令に続くユーザ定義パラメータをロードする。ＲＭＯＶ命令及びリマッピング・パラメータ・フォーマットの詳細については前記を参照。

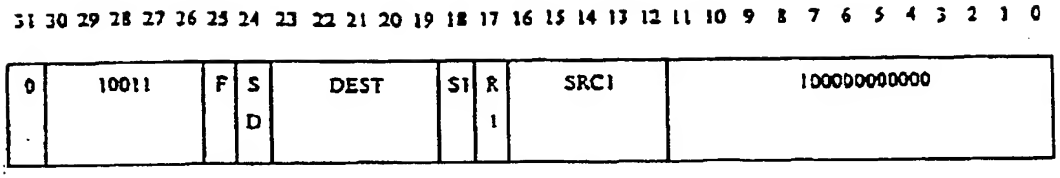
ループの繰り返し(iteration)の回数が０であれば、ＲＥＰＥＡＴの動作はＵＮＰＲＥＤＩＣＡＴＡＢＬＥ（予想不可）である。

命令数領域が０にセットされると、ＲＥＰＥＡＴの動作は、予想不可である。

ループに１つの命令しかなく、その命令がブランチであれば、予想不可能の挙動をする。

ＲＥＰＥＡＴループの範囲からそのループの外へのブランチは、予想不可である。

飽和絶対命令は、ソース１の飽和絶対値（saturated absolute）を計算する。



動作：

dest＝ＳＡＴ（(src１＞＝０)？src１：-src 1）．値は常に飽和する。特に、０ｘ８０００００００の絶対値は０ｘ７ｆｆｆｆｆｆｆであり、０ｘ８０００００００ではない。

ニューモニック：

ＳＡＢＳ＜dest＞，＜src１＞

フラグ

Ｚがセットされるのは、結果が０の時。

Ｎは保存される。

Ｃがセットされるのは、scr＜０（dest＝＿scr１の場合）。

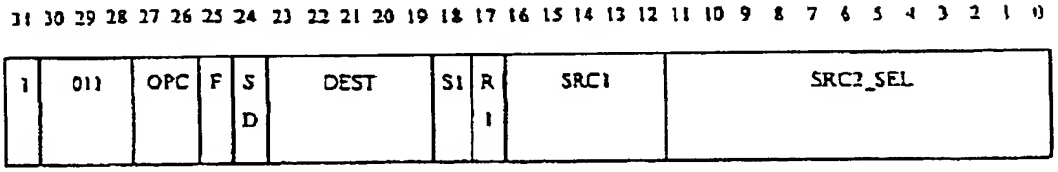
Ｖがセットされるのは、飽和が生じた時。

含む理由：

多くのＤＳＰアプリケーションで役立つ。

選択（select）操作（条件付き移動）は、条件付きでソース１またはソース２を目的レジスタに移動させる。選択は、常に、移動と等価である。並列加算／減算の後で使用するための並列操作もある。

尚、両方のソースオペランドは、導入理由のための命令によっても読み出すことができるので、一方が空きであれば、そのオペランドが絶対的に必要であるかどうかに関係なく、命令は止まる。



ＯＰＣは、命令のタイプを指定する。

動作（ＯＰＣ）：

00　If＜cond＞holds for primary flags　then dest=src1
（もし＜cond＞が一次フラグのためであれば）
else dest=src2.

01　If＜cond＞holds for the primary flags　then dest.h=src1.h
（もし＜cond＞が一次フラグのためであれば）
else dest.h=src2.h,
If＜cond＞holds for the secondary flags　then dest.l=src1.l
（もし＜cond＞が二次フラグのためであれば）
else dest.l=src2.l.

10　If＜cond＞holds for the primary flags　then dest.h=src1.h
（もし＜cond＞が一次フラグのためであれば）
else dest.h=src2.h,
If＜cond＞fails for the secondary flags　then dest.l=src1.l
（もし＜cond＞が二次フラグのためでなければ）
else dest.l=src2.l.

11　予約済

ニューモニック：

00	ＳＥＬ＜cond＞	＜dest＞, ＜src1＞, ＜src2＞
01	ＳＥＬＴＴ＜cond＞	＜dest＞, ＜src1＞, ＜src2＞
10	ＳＥＬＴＦ＜cond＞	＜dest＞, ＜src1＞, ＜src2＞
11	未使用	

レジスタが再充填の印になつていると、それは、無条件で再充填される。また、アセンブラ、次のニューモニックも提供する。

ＭＯＶ＜cond＞	＜dest＞, ＜src1＞
ＳＥＬＦＴ＜cond＞	＜dest＞, ＜src1＞, ＜src2＞
ＳＥＬＦＦ	＜cond＞＜dest＞, ＜src1＞, ＜src2＞

ＭＯＶ＜cond＞Ａ，Ｂは、ＳＥＬ＜cond＞Ａ，Ｂ，Ａと等価である。ＳＥＬＦＴとＳＥＬＦＦは、ＳＥＬＴＦ，ＳＥＬＴＴを使用して、src1とsrc2を交換す

ることによって得ることができる。

フラグ

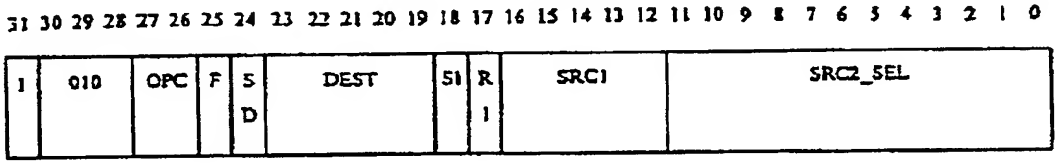
すべてのフラグは、一連の選択が行われるよう保存される。

含む理由：

簡単な決定をブランチに頼ることないインラインにするために使用される。最大要素のためにサンプル又はベクトルをスキャンする時に、そしてビタビ(Viterbi)アルゴリズムによって使用される。

シフト操作命令は、指定量の左右の論理シフト、右算術シフト、回転(rotate)を提供する。シフト量は、レジスタの内容の下８ビットから取られた－１２８から＋１２７の間の符号付き整数、又は、＋１から＋３１の範囲内の即値である。負の量のシフトは、ＡＢＳ（シフト量）分反対方向にシフトさせる。

入力オペランドは、３２ビットに符号拡張され、結果の３２ビット出力は、書き戻し前に４８ビットに符号拡張され、４８ビットレジスタへの書き込みが感度よく機能する。



ＯＰＣは、命令のタイプを指定する。

動作（ＯＰＣ）：

00　dest = (src2>=0) ? src1 <<src2 : src1 >>-src2

01　dest = (src2>=0) ? src1 >>src2 : src1 <<-src2

10　dest = (src2>=0) ? src1 - >>src2 : src1 <<-src2

11　dest = (src2>=0) ? src1 ROR src2 : src1 ROL -src2

ニューモニック：

00　ＡＳＬ　＜dest＞，＜src1＞，＜src2_16＞

01　ＬＳＲ　＜dest＞，＜src1＞，＜src2_16＞

10　ＡＳＲ　＜dest＞，＜src1＞，＜src2_16＞

11　ＲＯＲ　＜dest＞，＜src1＞，＜src2_16＞

フラグ

Ｚがセットされるのは、結果が０の時。

Ｎがセットされるのは、結果が負の時。

Ｖは保存される。

Ｃは、最後にシフトされて出た（ＡＲＭ上として）ビット値にセットされる。

レジスタ指定されたシフトの挙動は以下の通り。

－３２によるＬＳＬの結果は０で、src１のビット０にＣがセットされる。

－３２を越えるものでのＬＳＬは、結果が０で、Ｃは０にセットされる。

－３２によるＬＳＲの結果は０で、src１のビット３１にＣがセットされる。

－３２を越えるものでのＬＳＲは、結果が０で、Ｃは０にセットされる。

－３２以上でのＡＳＲの結果は充填され、Ｃはsrc１のビット３１に等しい。

－３２でのＲＯＲの結果はsrc１に等しく、Ｃがsrc１のビット３１にセットされる。

－３２を越えるｎによるＲＯＲは、ｎ－３２によるＲＯＲと同じ結果とキャリーアウト(carry out)になるので、量が１から３２の範囲内になるまで、繰り返し３２をｎから引く。上記参照。

含む理由：

２のべき乗による乗算／除算。ビット及び領域抽出。シリアル・レジスタ。

未定義の命令が、上記命令セットリストで挙げてある。それらの実行により、ピッコロは、実行を停止し、状態レジスタにＵビットをセットし、それ自身をディスエーブルする（制御レジスタ内のＥビットがクリアされたかのように）。これにより、命令が将来拡張された場合も、それがトラップされて、オプショナルに、既存の手段上でエミュレートされることが可能である。

ＡＲＭからピッコロ状態へのアクセスは以下の通り。状態アクセス・モードを

使用して、ピッコロの状態を観察／変更する。このメカニズムが提供されるのは次の２つの理由からである。

－文脈(Context)切替え

－デバッグ

ピッコロは、ＰＳＴＡＴＥ命令を行うことで、状態アクセスモードになる。このモードでは、ピッコロの状態を退避して、一連のＳＴＣ及びＬＤＣ命令で復元

される。状態アクセスモードに入ると、ピッコロ・コプロセッサＩＤ　ＰＩＣＣＯＬＯ１の使用が変更されて、ピッコロの状態にアクセスできるようになる。ピッコロの状態には７つのバンクがある。特定バンク内のすべてのデータは、単一のＬＤＣ又はＳＴＣでロードし記憶することができる。

バンク０：プライベート・レジスタ

－　ピッコロＩＤレジスタ(Read Only)の値を含む１つの３２ビットワード

－　制御レジスタの状態を含む１つの３２ビットワード

－　状態レジスタの状態を含む１つの３２ビットワード

－　プログラム・カウンタの状態を含む１つの３２ビットワード

バンク１：汎用レジスタ（ＧＰＲ）

－　汎用レジスタの状態を含む１６個の３２ビットワード

バンク２：アキュミュレータ

－　アキュミュレータ・レジスタの上の３２ビットを含む４つの３２ビットワード（注：ＧＰＲ状態の複製が復元に必要だということは、さもないとレジスタバンク上で別の書き込みイネーブルを意味する）。

バンク３：レジスタ／ピッコロＲＯＢ／出力ＦＩＦＯ状態

－　どのレジスタが再充填用の印（各３２ビットレジスタにつき２ビット）になっているかを示す３２ビットワードが１つ。

－　ＲＯＢタグ（ビット７から０に記憶されている７ビット項目８つ）の状態を含む３２ビットワード８つ。

－　連合していない(unaligned)ＲＯＢラッチ（ビット１７から０）の状態を含む３２ビットワード３つ。

－　出力シフトレジスタ内のどのスロットが有効データを含むかを示す３２ビットワードが１つ（ビット４は空きを示し、ビット３から０は、使用中のエントリの数をエンコードする）。

－　ラッチ（ビット１７から０）を保持する出力ＦＩＦＯの状態を含む３２ビットワード１つ。

バンク４：ＲＯＢ入力データ

－　３２ビットデータ値が８つ。

バンク５：出力ＦＩＦＯデータ

－　３２ビットデータ値が８つ。

バンク６：ループハードウェア

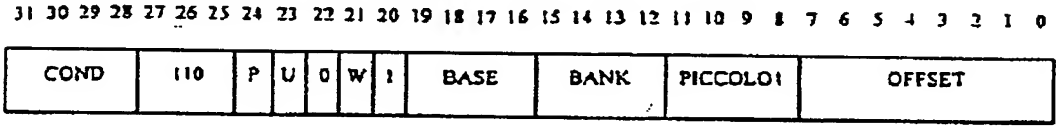
－　ループ開始アドレスを含む３２ビットワード４つ。

－　ループ最終アドレスを含む３２ビットワード４つ。

－　ループ回数（ビット１５から０）を含む３２ビットワード４つ。

－　ユーザ定義リマッピング・パラメータその他のリマッピング状態を含む３２ビットワードが１つ。

ＬＤＣ命令は、ピッコロが状態アクセスモードにある時にピッコロの状態をロードするのに使う。ＢＡＮＫ領域はロードされるバンクを特定する。



次の一連の動作により、ピッコロのすべての状態がレジスタＲ０内のアドレスからロードされる。

ＬＤＰ　Ｂ０，［Ｒ０］，＃１６！；プライベート・レジスタ

ＬＤＰ　Ｂ１，［Ｒ０］，＃６４！；汎用レジスタをロードする。

ＬＤＰ　Ｂ２，［Ｒ０］，＃１６！；アキュミュレータをロードする。

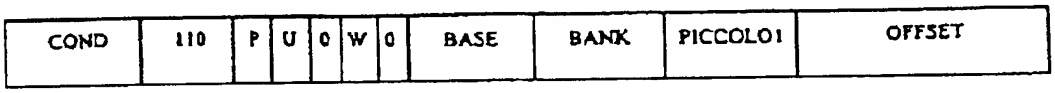
ＬＤＰ　Ｂ３，［Ｒ０］，＃５６！；レジスタ／ＲＯＢ／ＦＩＦＯ状態をロードする。

ＬＤＰ　Ｂ４，［Ｒ０］，＃３２！；ＲＯＢデータをロードする。

ＬＤＰ　Ｂ５，［Ｒ０］，＃３２！；出力ＦＩＦＯをロードする。

ＬＤＰ　Ｂ６，［Ｒ０］，＃５２！；ループハードウェアをロードする。

ＳＴＣ命令は、ピッコロが状態アクセスモードにある時にピッコロの状態を記憶させるのに使う。ＢＡＮＫ領域はどのバンクが記憶されるかを特定する。



次の一連の動作により、ピッコロのすべての状態がレジスタＲ０内のアドレスから記憶される。

ＳＴＰ　Ｂ０，［Ｒ０］，＃１６！；プライベートレジスタを退避する。

ＳＴＰ　Ｂ１，［Ｒ０］，＃６４！；汎用レジスタを退避する。

ＳＴＰ　Ｂ２，［Ｒ０］，＃１６！；アキュミュレータを退避する。

ＳＴＰ　Ｂ３，［Ｒ０］，＃５６！；レジスタ／ＲＯＢ／ＦＩＦＯ状態を退避する。

ＳＴＰ　Ｂ４，［Ｒ０］，＃３２！；ＲＯＢデータを退避する。

ＳＴＰ　Ｂ５，［Ｒ０］，＃３２！；出力ＦＩＦＯを退避する。

ＳＴＰ　Ｂ６，［Ｒ０］，＃５２！；ループハードウェアを退避する。

デバッグ・モード　－　ピッコロは、ＡＲＭによってサポートされているものと同じデバッグ・メカニズム、すなわち、DemonとAngelを介したソフトウェア、及び埋め込まれたＩＣＥを備えたハードエウェア、に応答しなければならない。ピッコロのシステムをデバッグするためのいくつかのメカニズムがある。

－ＡＲＭ命令ブレークポイント

－データ・ブレークポイント（ウオッチポイント）

－ピッコロ命令ブレークポイント

－ピッコロ・ソフトウェア・ブレークポイント

ＡＲＭ命令ブレークポイント及びデータ・ブレークポイントは、ＡＲＭ埋め込みＩＣＥモジュールによって扱われる。ピッコロ命令ブレークポイントは、ピッ

コロ埋め込みＩＣＥモジュールによって扱われる。ピッコロ・ソフトウェア・ブレークポイントは、ピッコロ・コアによって扱われる。ハードウェア・ブレークポイント・システムは、ＡＲＭとピッコロの両方がブレークポイントされるように構成される。

ソフトウェア・ブレークポイントを扱うのは、ピッコロ命令（Halt又はBreak）で、ピッコロに実行を止めさせ、デバッグ・モードに入れ（状態レジスタのＢビットがセットされる）、自身をディスエーブルする（ピッコロがＰＤＩＳＡＢＬＥ命令によってディスエーブルされたようになる）。プログラム・カウンタは有効のままで、ブレークポイントのアドレスが回復できる。ピッコロは、それ以上、命令を実行しなくなる。

Single stepping Piccoloは、ピッコロ命令ストリーム上に次々にブレークポイントをセットすることによって行われる。

ソフトウェア・デバッグ　－　ピッコロによって提供される基本的機能は、状態アクセスモードにある時、コプロセッサ命令を介して、すべての状態をメモリーにロード及び退避させる能力である。これにより、デバッガーは、すべての状態をメモリーに退避させ、それを読み出し，及び／又は更新し、それをピッコロに復元することができる。ピッコロの記憶状態メカニズムは、非破壊的であり、つまり、ピッコロの状態を記憶する動作は、ピッコロの内部状態を駄目にすることはない。つまり、ピッコロは、その状態をダンプした後、それを復元することなしに、再開できる。

ピッコロ・キャッシュの状態を見つけるメカニズムを決定しなければならない。

ハードウェア・デバッグ　－　ハードウェア・デバッグは、ピッコロのコプロセッサ・インターフェース上のスキャン・チェインによって行うことができる。ピッコロは状態アクセスモードになり、スキャン・チェインを介して、その状態を調査／変更してもらう。

ピッコロの状態レジスタは、ブレークポイント付き命令を実行したことを示す単一ビットを含む。ブレークポイント付き命令が実行されると、ピッコロは、状

態レジスタにＢビットをセットし、実行を中止する。ピッコロに質問をするには、デバッガーは、ピッコロをイネーブルし、次のアクセスが起きる前に、制御レジスタに書き込むことによって、状態アクセスモードにしなければならない。

図４は、Ｈｉ／ＬｏビットとSizeビットに応答して、選択されたレジスタの適当な半分をピッコロ・データパスに切り換えるマルチプレクサ構成を示す。Sizeビットが１６ビットなら、符号拡張回路が必要に応じてデータパスの高次ビットに０か１を入れる。

セクション２

図８は、図１のシステムを変更し発展させたものを示す。中央処理装置（以下、ＣＰＵとする）コア２は、命令デコーダ８０と、命令パイプライン８２と、レジスタバンク８４と、メモリコントローラ８６とを備える。作動中、ＣＰＵ命令は、メモリコントローラ８６によってメモリ８から引き出され、命令パイプライン８

２に供給される。命令は、命令パイプラインに沿って、命令デコーダ８０に隣接する段に到達する。この段において、命令を実行するためのデコーディングは完結する。命令デコーダ８０は、命令内のビットフィールドに応答し、ＣＰＵコア２の他の部分を構成し駆動する制御信号を提供する論理回路を使用して、所望のデータ処理動作を行う。実際には、ＣＰＵコア２は、多数の機能ブロック、例えば、演算ユニット、マルチプライア、キャッシュ、メモリ管理ユニットを備える。

本例では、コプロセッサメモリアクセス命令が命令デコーダ８０によってデコードされる時、メモリ８内のメモリ位置を指すアドレスポインタとなるアドレス値を保持するレジスタバンク８４内のレジスタが指定される。このアドレス値は、アドレスバスに出されメモリ８に到達し、メモリコントローラを介して、バーストモード転送を開始する。転送されるデータは、ＣＰＵコア２よりも、むしろコプロセッサ４に向けられる。従って、メモリ８に対して適切なアドレスを生成する以外は、ＣＰＵコア２は、メモリ８とコプロセッサ４との間のデータバス上にあるデータワードに応答しない。命令デコーダ８０も、コプロセッサ制御信号

（ＣＰ　Ｃｏｎｔｒｏｌ）を生成し、それらはコプロセッサ４に渡される。これらのコプロセッサ制御信号は、コプロセッサに対して、コプロセッサメモリアクセス命令が実行中であることを示し、従って、コプロセッサ４は、データをデータバスに出すか、データバスからデータを読み取るか、適切な動作を行わなければならない。コプロセッサ４に渡されるコプロセッサ制御信号は、実行中のコプロセッサメモリアクセス命令内のアドレシングモード情報の少なくとも一部を含む。より詳しくは、少なくとも、Ｐフラグ、Ｕフラグ、及びオフセット値Ｍがコプロセッサ４に渡される。

コプロセッサ４は、排他的論理和（ＥＯＲ）演算をこれらのビットに行うことによって、ＰフラグとＵフラグをデコードする。この排他的論理和演算の結果によって、コプロセッサは、現在のバーストで転送されるべきデータワードの個数が、レジスタ８８に記憶されていて渡されるオフセット値Ｍによって指定されるか、１つのデータワードというデフォルト値にすべきかを決定する。コプロセッサ転送コントローラ９０は、レジスタ８８の出力と排他的論理和演算の結果に応答して、データバス上で受け取られたデータワードの個数を数え、指定された数

が受け取られ、バースト終了信号が出ると、それは、メモリ８とＣＰＵコア２に返されて、実行中のコプロセッサメモリアクセス命令によって開始された転送を終了する。コプロセッサ４によって、データバスから受け取られたデータワードは、どれも、リオーダ（ｒｅｏｒｄｅｒ）バッファ１２にロードされてから、コプロセッサコア９２によって処理される。あるいは、コプロセッサ４は、バーストの長さを直接メモリ８へ提供することもできる（これは、例えば、同期ＤＲＡＭのような型に有効である）。

図９は、上述のコプロセッサメモリアクセス命令の動作を模式的に示すものである。

動作は、ステップ９４で開始され、ステップ９６に移り、そこでＣＰＵがコプロセッサメモリアクセス命令に埋め込まれたアドレシングモード情報を読み取り、一方、コプロセッサは、この同じアドレシングモード情報の少なくとも一部を読み取り、そこから、転送されるデータワードの個数を決定する。

ステップ９８で、ＣＰＵは、メモリ８に提供されるアクセス開始アドレスを生成する。ステップ１００で、メモリ８とコプロセッサ４との間で直接的に、データワードが転送される。データワードが１つ転送されるごとに、ステップ１０２において、コプロセッサメモリアクセス命令により指定された個数のデータワードがすべて転送されたかどうかを、コプロセッサ４が決定する。

転送が完了しないうちは、ＣＰＵコア２の動作が継続し、ステップ１０４におけるアドレスに必要な更新を行い、ステップ１００に戻る。

転送が完了すると、システムはステップ１０６に進み、そこで、コプロセッサは、転送を終了するために、バースト完了信号を出して、それがメモリ８とＣＰＵコア２の両方に渡される。ステップ１０８において、ＣＰＵコア２は、コプロセッサメモリアクセス命令によって指定されたように、アドレスポインタを更新する（これは、プロセスの他の点において行ってもよい）。このプロセスは、ステップ１１０で終了する。

性能を改善するために、転送は、並行的に行ってもよい。例えば、コプロセッサがＣＰＵに対して、第１のワードの転送が開始する前でも、第２のデータワードが転送されるべきであるかどうかを指示することもできる。

以下に、本発明の実施の形態により動作する様々のコプロセッサメモリアクセス命令について詳細に述べる。転送されるデータワードの個数を制御するという視点から見ると、これらの命令の全体的な動作は、ＰフラグとＵフラグに従って、次のアドレス転送モードのどれかになる。

	転送開始アドレス値	アドレスレジスタ内の最終値	転送されるデータワード個数
(i)	Rn	Rn-(WL*M)	1
(ii)	Rn	Rn	M
(iii)	Rn	Rn+(WL*M)	M
(iv)	Rn-(WL*M)	Rn	M
(v)	Rn-(WL*M)	Rn-(WL*M)	M
(vi)	Rn+(WL*M)	Rn	1
(vii)	Rn+(WL*M)	Rn+(WL*M)	1

この他に、ベースレジスタがＡＲＭプログラムカウンタレジスタ（ＰＣ又はＲ１５）である場合の転送されるワードの個数がある。この場合、Ｐ　ＥＯＲ（Ｕ又は（ベースレジスタがＰＣである））への単一ワード転送を決定するロジックを変更することになる。

メモリからリオーダバッファへのローディング

フォーマットの要約

リオーダバッファにメモリからのデータを入れるのに２つの主要なフォーマットがある。

－ＬＤＰ

－ＬＰＭ

どちらの命令のフォーマットも、ＡＲＭ　ＬＤＣ命令としてエンコードされる。ＬＤＰ命令クラスは、常に、１つの３２ビットのデータワードをメモリから転送

する。ＬＰＭ命令クラスは、複数のワードを転送するのに使用することができる。命令ビットパタンのレベルでは、ＬＤＰとＬＰＭは、使用されるアドレシングモードビットによって区別され、アセンブラーの文法は、ＬＤＰとＬＰＭに対して異なったニューモニックを使用し、１つのワードか２つ以上のワードの転送か

についてのコードを書く又は読む人に気付かせる。この２つの命令に対して、以下のフォーマットが使用できる。

LDP{cond}[32|16]	dest, address
LDP{cond}16U	bank, address
LPM{cond}[A|D][32|16]	Rn{!}, [<Rlist>]{,#<wordcount>}
LPM{cond}[A|D]16	Rn{!}, <bank0.1>, #<wordcount>

ここにおいて、

{}　　は、オプションフィールドを示す。

ｃｏｎｄ　　は、ＡＲＭ命令条件コードフィールドである。

３２｜１６　　は、ロードされているデータが１６ビットデータとして扱われエンディアネス（ｅｎｄｉａｎｅｓｓ）用アクションを取る（ＳＴＰ１６及びＳＴＰ３２を参照）べきか、あるいは３２ビットデータとして扱われるべきかを示す。

ｄｅｓｔ　　は、ピッコロ目的レジスタ（Ａ０乃至Ｚ３）を指定する。

ａｄｄｒｅｓｓは、以下のいずれかである。

〔Ｒｎ〕

〔Ｒｎ，＃＋ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

〔Ｒｎ〕＃＋ｖｅ＿ｏｆｆｓｅｔ

Ｒｎ　　は、有効なＡＲＭレジスタ番号を求める表現である。

！　　は、計算されたアドレスがベースレジスタに書き戻す必要があることを示す。

＃＋ｖｅ＿ｏｆｆｓｅｔ　は、＋＜８－ｂｉｔ－ｏｆｆｓｅｔ＞＊４として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに加算されロードアドレスを形成する。即ち、アドレスが前もってインデクスをつけられる。

＃－ｖｅ＿ｏｆｆｓｅｔ　は、－＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞＊４として表すことのできるオフセットを求める表現である。このロードアドレスは、ベースレジスタＲｎの値であり、オフセット分がＲｎから減算され、結果がＲｎに書き戻

される。

ｂａｎｋ　　は、３つの非蓄積型ピッコロバンク（Ｘ｜Ｙ｜Ｚ）の１つを指定する。

Ａ｜Ｄ　　は、プレインクリメント（上り）、又は、ポストデクリメント（下り）アドレシングモードを指定する。

＜Ｒｌｉｓｔ＞は、ピッコロレジスタ番号の上りリストであり、例えば、｛Ｘ０，Ｘ１｝というように、｛｝に挟まれている。ピッコロの第１版では、最大８つのレジスタを指定することができる。このリストは、レジスタバンクのトップで巡回しても良い。例えば、｛Ｙ２，Ｙ３，Ａ０，Ａ１｝は、有効なレジスタリストである。

ｂａｎｋ０．１　は、４個の１６ビットレジスタ（Ａ０．１｜Ｘ０．１｜Ｙ０．１｜Ｚ０．１）の１つを指定する。

ｗｏｒｄｃｏｕｎｔ　は、選択された領域のレジスタ上でラッピング（ｗｒａｐｐｉｎｇ）ロードするのに使用され、転送されるデータ項目の総数を指定する。ピッコロの第１版では、１つのＬＰＭ命令で最大８つまでのデータ項目を転送することができる。

＜ｌｉｓｔ＞フォーマットが使用される場合、＜ｌｉｓｔ＞によって指定されるレジスタのリストは、（Ａ０，Ｘ０，Ｙ０，Ｚ０）のいずれかで始めなければならず、レジスタリストは、１個、２個、又は４個のレジスタを指定することができる。即ち、レジスタのＸバンクに対して、

｛Ｘ０｝

｛Ｘ０，Ｘ１｝

｛Ｘ０，Ｘ１，Ｘ２，Ｘ３｝

だけが、有効な＜Ｒｌｉｓｔ＞組み合わせである。＜ｗｏｒｄｃｏｕｎｔ＞は、（Ｒｌｉｓｔ）の長さより大きくなければならない。このＬＰＭ命令のフォーマットは、＜ｗｏｒｄｃｏｕｎｔ＞データ項目全部を転送する。その際、

＜Ｒｌｉｓｔ＞内の各レジスタに対して、リストの最後に来たら、

＜Ｒｌｉｓｔ＞の開始へ巡回するようにデータにタグをつける。

＜ｂａｎｋ０．１＞フォーマットが使用される場合は、ピッコロの第１版では、＜ｗｏｒｄｃｏｕｎｔ＞は、（１－８）の範囲が可能である。このＬＰＭ命令のフォーマットは、２＊＜ｗｏｒｄｃｏｕｎｔ＞１６ビットのデータ項目をロードし、その際、すべてのデータに、レジスタ＜ｂａｎｋ０．１＞用タグをつける。

例

ＬＤＰＮＥ３２　Ａ１，〔Ｒ０〕：Ｚフラグ＝０であれば、Ａ１にｍｅｍ（Ｒ０）をロードする。データを３２ビットとして扱う。

ＬＤＰ１６　Ｘ０，〔Ｒ０，＃１６〕！：Ｘ０にｍｅｍ（Ｒ０＋１６（バイト））をロードし、Ｒ０＋１６をＲ０に書き戻す。データをパックト（ｐａｃｋｅｄ）１６ビットとして扱う。

ＬＤＰ３２　Ｙ２，〔Ｒ０〕，＃－４：Ｙ２にｍｅｍ（Ｒ０）をロードし、Ｒ０に（Ｒ０－＃４）を書き戻す。

ＬＤＰ１６Ｕ　Ｘ，〔Ｒ０，＃４〕：Ｘバンクの非配列ラッチにｍｅｍ（Ｒ０＋＃４（バイト））を入れる。

ＬＰＭＥＱＡ３２　Ｒ０！，｛Ｘ２，Ｘ３，Ｙ０，Ｙ１｝：Ｚフラグ＝１であれば、４ワードを上り順で、メモリ〔Ｒ０〕からロードする。その際、レジスタＸ２，Ｘ３，Ｙ０，Ｙ１用にタグをつける。Ｒ０に書き戻す。

ＬＰＭＡ１６　Ｒ１！，｛Ｘ０，Ｘ１｝，＃８：パックト１６ビットデータ８ワードを、上り順で〔Ｒ１〕のメモリからロードする。その際、レジスタＸ０，Ｘ１，Ｘ０，Ｘ１，Ｘ０，Ｘ１，Ｘ０，Ｘ１用にタグをつける。

命令エンコーディング

ＬＤＰ命令

ＬＤＰ命令は、１つの３２ビットワードをメモリから転送する。これらの命令のいくつかは、書き戻しを行うが、アセンブラの文法に従って、「！」の印がついていない。なぜなら、ポスト・インデクシングは常に書き戻しを含むからである。２つの変形がある。

ＬＤＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ〕，＃－ｖｅ＿

ｏｆｆｓｅｔ

ＬＤＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ，＃－ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

アドレシングモードは、Ｐ，Ｕ，Ｗビットによって決まる。これらは、それぞれ命令内のビット２４、２３、２１にある。

Ｐ＝０，Ｕ＝０，Ｗ＝１という組み合わせは、次の形式の命令をエンコードするのに使用される。

ＬＤＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ

アドレス〔Ｒｎ〕からは、ワードが１つだけ転送される。転送が行われた後、ベースレジスタは、８＿ｂｉｔ＿ｏｆｆｓｅｔ＊４だけデクリメントされる。Ｎビットは、ＬＤＰ３２（１）またはＬＤＰ１６（０）を指定する。＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、ｍｏｄ（＃－ｖｅ＿ｏｆｆｓｅｔ）／４をエンコードする。この命令は、データ構造を介して逆戻りして１つの特定のワードを出現毎に抽出するのに役立つ。同様の所望の機能は、他の命令とも合わせられる。

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-------+-----+---+-+---+-------+-------+-------+---------------+
| cond  |1 1 0|0 0|N|1 1|  Rn   | dest  | pic_1 |  8_bit_offset |
+-------+-----+---+-+---+-------+-------+-------+---------------+
```

Ｐ＝１，Ｕ＝１の組み合わせは、次の形式の命令をエンコードするのに使用される。

ＬＤＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ，＃－ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

アドレス〔Ｒｎ＃－ｖｅ＿ｏｆｆｓｅｔ〕からはワードが１つだけ転送される。Ｗビットがセットされると、ベースレジスタは、８＿ｂｉｔ＿ｏｆｆｓｅｔ＊４だけインクリメントされる。即ち、オプションの「！」により書き戻しが指定されている形式である。Ｗ＝０の場合は、書き戻しは行われず、「！」もない。

Ｎ

ビットは、ＬＤＰ３２（１）又はＬＤＰ１６（０）を指定する。＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、＃＋ｖｅ＿ｏｆｆｓｅｔ／４をエンコードする。Ｐｉｃ＿１（及び，後で述べるＰｉｃ＿２）は、コプロセッサがピッコロ・コプロセッサであることを示す、識別番号である。ピッコロは、関連命令に従って使用される２つの識別番号を持っている。

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-------+-----+---+-+-+-------+------+------+---------------+
| cond  |1 1 0|1 1|N|W|1|  Rn   | dest | pic_1 |  8_bit_offset |
+-------+-----+---+-+-+-------+------+------+---------------+
```

ＬＤＰ１６Ｕ

ＬＤＰ１６Ｕ命令は、３つの非配列保持ラッチの１つを用意するのに使用される。それは、次の変形を持つ。

ＬＤＰ｛ｃｏｎｄ｝Ｕ１６　ｂａｎｋ，〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ

ＬＤＰ｛ｃｏｎｄ｝Ｕ１６　ｂａｎｋ，〔Ｒｎ，＃＋ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

アドレシングモードはＰ及びＵビットにより決まる。

Ｐ＝０，Ｕ＝０，Ｗ＝１という組み合わせは、次の形式の命令をエンコードするのに使用される。

ＬＤＰ｛ｃｏｎｄ｝Ｕ１６　ｄｅｓｔ，〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ

アドレス〔Ｒｎ〕から転送されるワードは１つだけである。この転送の後、ベースレジスタは、＃－ｖｅ＿ｏｆｆｓｅｔにより変更される。

```
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-------+-----+---+-+-+-------+---+---+-------+---------------+
| cond  |1 1 0|0 0|0|1|1|  Rn   |bnk|0 0| pic_2 |  8_bit_offset |
+-------+-----+---+-+-+-------+---+---+-------+---------------+
```

＜ｂｎｋ＞は、非配列モードにするバンクを指定し、バンクＸ，Ｙ，Ｚに対して

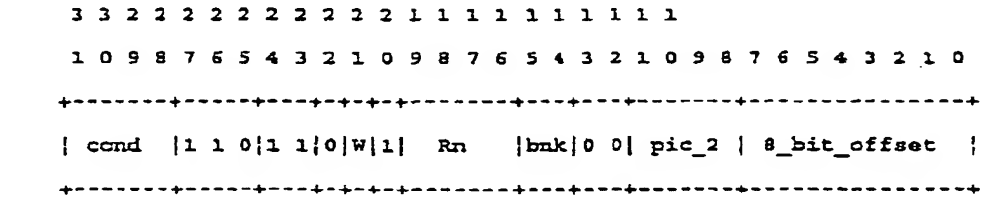
、

１、２、３の値を取ることができる。

Ｐ＝１、Ｕ＝１の組み合わせは、次の形式の命令をエンコードするのに使用される。

ＬＤＰ｛Ｃｏｎｄ｝Ｕ１６　ｄｅｓｔ，〔Ｒｎ，＃－ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

アドレス〔Ｒｎ＋＃＋ｖｅ＿ｏｆｆｓｅｔ〕から転送されるワードは１つだけである。Ｗビットがセットされると、ベースレジスタは、＃＋ｖｅ＿ｏｆｆｓｅｔにより変更される。Ｗ＝０なら、書き戻しはない。



＜ｂｎｋ＞は、非配列モードにするバンクを指定し、バンクＸ，Ｙ，Ｚに対して、１、２、３の値を取ることができる。

ＬＰＭ命令

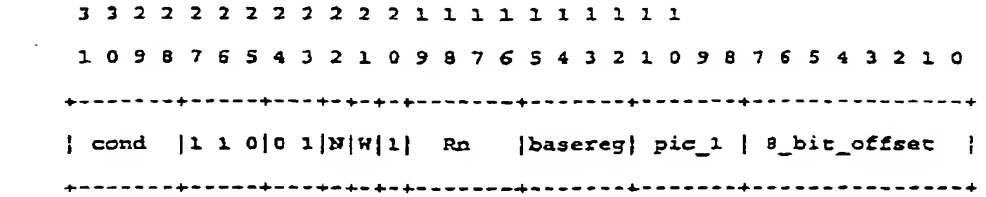
ＬＰＭ命令は、メモリから２つ以上のワードを転送し、次の変形がある。

LPM{cond}[A|D][32|16]　Rn{!}, [<Rlist>]

LPM{cond}[A|D][32|16]　Rn{!}, [<Rlist>], #<wordcount>

LPM{cond}[A|D]16　Rn{!}, <bank0.1>, #<wordcount>

ＬＰＭＡ形に対してはＰ＝０．Ｕ＝１であり、命令は、次のようにエンコードされる。



ここにおいて、

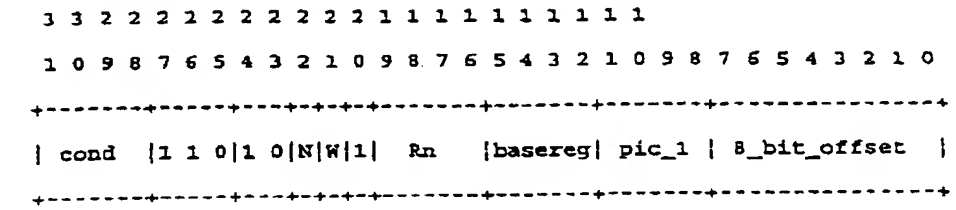
Ｎビットは、ＬＰＭＡ３２（１）又はＬＰＭＡ１６（０）を指定する。

Ｗビットは、Ｗ＝１なら、ベースレジスタへのｂａｓｅｒｅｇ＝ｏｆｆｓｅｔ

*４の書き戻しを指定する。

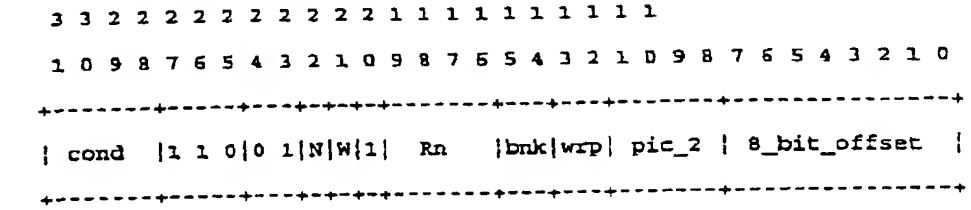
＜ｂａｓｅｒｅｇ＞は、＜Ｒｌｉｓｔ＞内の第１ピッコロレジスタを指定する。

＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、転送されるべきレジスタの個数を指定する。ＬＰＭＤ形に対してはＰ＝１．Ｕ＝０であり、命令は、次のようにエンコードされる。



LPM{cond}[A|D][32|16]　Rn{!}, ＜Rlist＞, #＜wordcount＞

ＬＰＭＡ形に対してはＰ＝０、Ｕ＝１であり、命令（この場合も、以下においても、ｐｉｃ－２，ｂｎｋ　ｗｒｐフォーマットを使用する）は、次のようにエンコードされる。



ここにおいて、

Ｎビットは、ＬＰＭＤ３２（１）又はＬＰＭＤ１６（０）を指定する。

Ｗビットは、Ｗ＝１なら、ベースレジスタへのｂａｓｅｒｅｇ＋ｏｆｆｓｅｔ*４の書き戻しを指定する。

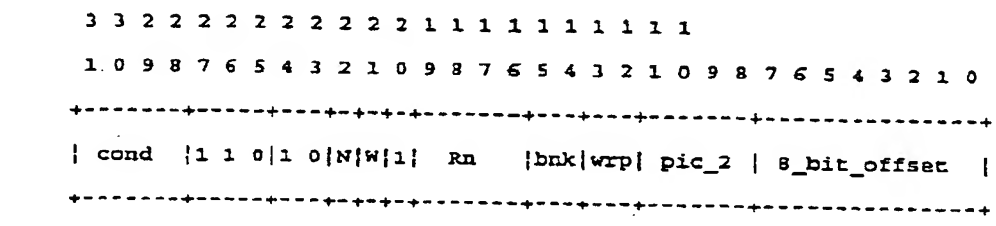
＜ｂｎｋ＞は、＜Ｒｌｉｓｔ＞内の第１レジスタを指定し、それがバンクのベースでなければならない（即ち、Ａ０，Ｘ０，Ｙ０，又はＺ０）。それは、バンクＡ－Ｚを指示する値０－３を取ることができる。

＜ｗｒｐ＞は、ラッビング（ｗｒａｐｐｉｎｇ）点を指定し、ラップ値として１－３の値を取ることができ、それぞれ、２個、４個、８個の１６ビットレジス

タを表す。

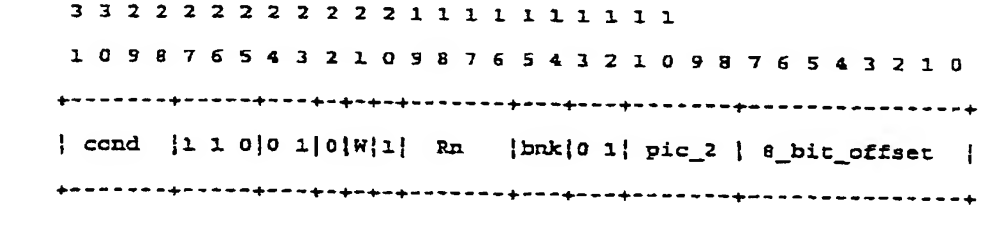
＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、転送すべきデータの個数を指定する。それは、＜ｗｏｒｄｃｏｕｎｔ＞／４の値を取る。

ＬＰＭＤ形に対してはＰ＝１、Ｕ＝０であり、命令は、次のようにエンコードされる。



LPM{cond}[A|D]16　　Rn{!}, <bank0.1>, #<wordcount>

ＬＰＭＡ形に対してはＰ＝０、Ｕ＝１であり、命令は、次のようにエンコードされる。



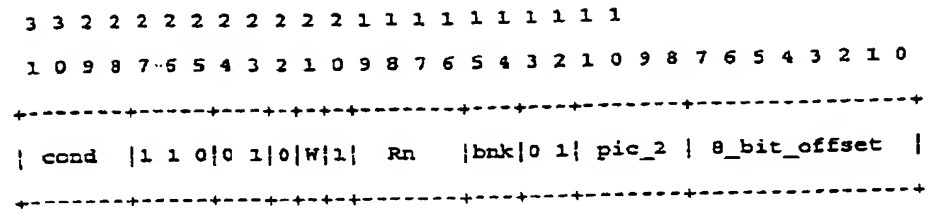
ここにおいて、

Ｗビットは、ベースレジスタ（１）書き戻しを指定する。

＜ｂｎｋ＞は、転送先のピッコロバンクを指定し、Ａ，Ｘ，Ｙ，又はＺを表す値０－３を取ることができる。

＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、転送すべきデータ項目の個数を指定する。それは、＜ｗｏｒｄｃｏｕｎｔ＞／４の値を取る。

ＬＰＭＤ形に対してはＰ＝０、Ｕ＝１であり、命令は、次のようにエンコードされる。



ＡＲＭレジスタからのデータのリオーダバッファへのローディング

フォーマットの要約

ＡＲＭレジスタからピッコロ・リオーダバッファへデータワードを転送するためのＭＰＲ命令フォーマットがある。ＭＰＲ命令には、以下のフォーマットが使用できる。

ＭＰＲ｛ｃｏｎｄ｝　ｄｅｓｔ，Ｒｎ

ＭＰＲ｛ｃｏｎｄ｝Ｗ　ｄｅｓｔ，Ｒｎ

ここで、

｛｝　　は、オプションフィールドを示す。

ｃｏｎｄ　は、ＡＲＭ命令条件コードフィールドである。

ｄｅｓｔ　は、ピッコロ目的レジスタ（Ａ０－Ｚ３）を指定する。

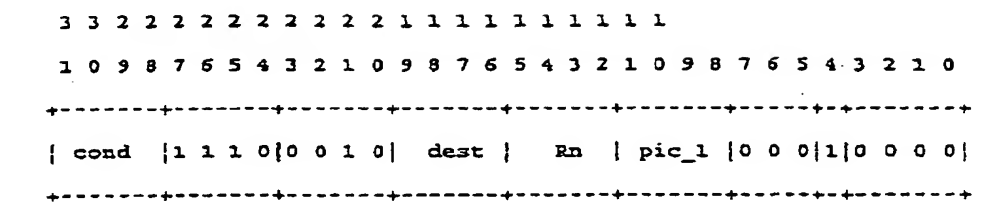
Ｒｎ　　は、有効なＡＲＭレジスタ番号を求める表現である。

Ｗ　　は、ＡＲＭレジスタから転送されるデータが、２個の１６ビット値として扱われ、ピッコロレジスタｄｅｓｔ０．１用のタグをつけなければならないことを示す。

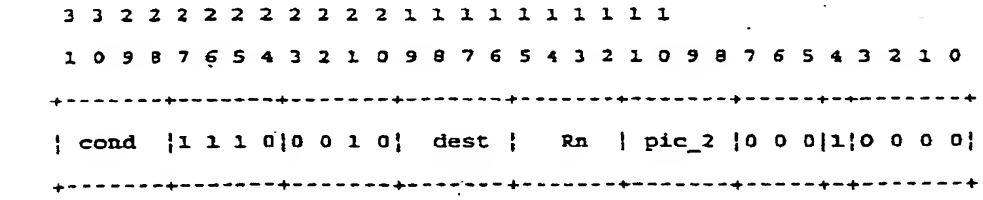
命令コーディング

ＭＰＲ命令は、ＡＲＭ　ＭＣＲ命令としてエンコードされる。

MPR{cond}　dest, Rn



MPR{cond}W　dest, Rn



出力ＦＩＦＯからメモリへのデータ記憶

フォーマットの要約

出力ＦＩＦＯからメモリへのデータ記憶には、２つの主要命令がある。

－ＳＴＰ

－ＳＰＭ

どちらの命令も、ＡＲＭ　ＳＴＣ命令としてエンコードされる。ＳＴＰ命令クラスは、常に、１つの３２ビットのデータワードを出力ＦＩＦＯからメモリへ記憶する。ＳＴＭ命令クラスは、２つ以上のワードを記憶するのに使用される。これら２つの命令には、次のフォーマットが使える。

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕　ａｄｄｒｅｓｓ

ＳＴＭ｛ｃｏｎｄ｝〔Ａ｜Ｄ〕〔３２｜１６〕Ｒｎ｛！｝，

＃＜ｗｏｒｄｃｏｕｎｔ＞

ここで、

｛｝　　は、オプションフィールドを示す。

ｃｏｎｄ　は、ＡＲＭ命令条件コードフィールドである。

３２｜１６は、記憶されるデータが１６ビットデータとして扱われ、（前述された）エンディアネス用動作が行われるべきか、３２ビットデータとして扱われるべきかを示す。

ａｄｄｒｅｓｓ　は、次のいずれかになる。

〔Ｒｎ〕

〔Ｒｎ，＃＋ｖｅ＿ｏｆｆｓｅｔ〕｛！｝

〔Ｒｎ〕，＃＋ｖｅ＿ｏｆｆｓｅｔ

Ｒｎ　　は、有効なＡＲＭレジスタ番号を求める表現である。

！　　は、計算されたアドレスがベースレジスタに書き戻されなければならないことを示す。

＃＋ｖｅ＿ｏｆｆｓｅｔ　は、＋＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞＊４として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに加算されて、記憶アドレスを形成する。

＃－ｖｅ＿ｏｆｆｓｅｔ　は、－＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞＊４として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに減算されて、ポストストアアドレスを形成する。

Ａ｜Ｄ　　は、プレインクリメント（上り）又はポストデクリメント（下り）アドレシングモードを示す。

ｗｏｒｄｃｏｕｎｔ　は、転送されるデータの総数項目を示す。ピッコロの第１版では、１つのＳＰＭ命令で、最大８個のデータ項目を転送できる。

命令エンコーディング

ＳＴＰ命令

ＳＴＰ命令は、１個の３２ビットワードをメモリに転送する。２つの変形がある。

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕　ｄｅｓｔ，〔Ｒｎ，＃＋ｖｅ＿ｏｆｆｓｅ

ｔ〕｛！｝

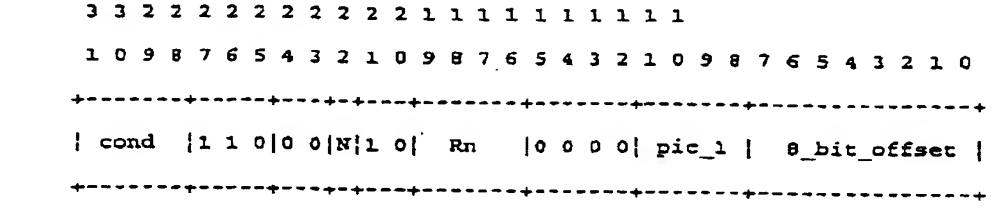
アドレシングモードは、Ｐ、Ｕビットにより決まる。

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕　〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ（Ｐ＝０｜Ｕ＝０｜Ｗ＝１）

Ｐ＝０，Ｕ＝０，Ｗ＝１の組み合わせは、次の形式の命令をエンコードするのに使用される。

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕　〔Ｒｎ〕，＃－ｖｅ＿ｏｆｆｓｅｔ

アドレス〔Ｒｎ〕に転送されるワードは、１個だけである。転送が行われた後、ベースレジスタは、８＿ｂｉｔ＿ｏｆｆｓｅｔ*４だけデクリメントされる。Ｎビットは、ＳＴＰ３２（１）又はＳＴＰ１６（０）を指定する。Ｗ＝０のエンコーディングは許可されない。＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、ｍｏｄ（＃－ｖｅ＿ｏｆｆｓｅｔ）／４をエンコードする。



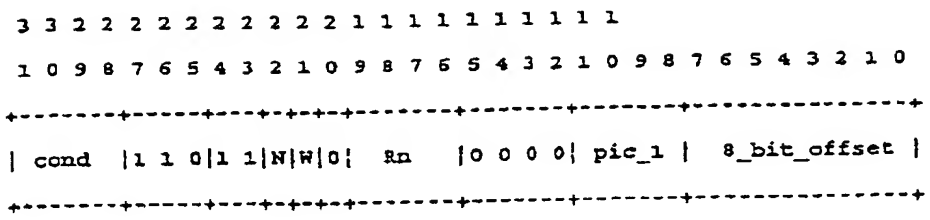
STP{cond}[32|16]　　dest, [Rn, #+ve_offset]{!} (P=1|U=1)

Ｐ＝１，Ｕ＝１の組み合わせは、次の形式の命令をエンコードするのに使用される。

ＳＴＰ｛ｃｏｎｄ｝〔３２｜１６〕ｄｅｓｔ，〔Ｒｎ，＃＋ｖｅ＿

ｏｆｆｓｅｔ〕｛！｝

アドレス〔Ｒｎ，＃＋ｖｅ＿ｏｆｆｓｅｔ〕に転送されるワードは１つだけである。Ｗビットがセットされると、ベースレジスタは、８＿ｂｉｔ＿ｏｆｆｓｅｔ*４だけインクリメントされる。Ｎビットは、ＳＴＰ（１）又はＳＴＰ１６（０）を指定する。＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、＃＋ｖｅ＿ｏｆｆｓｅｔ／４をエンコードする。



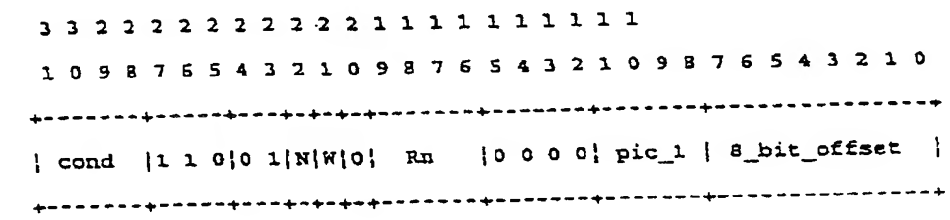
ＳＰＭ命令

ＳＰＭ命令は、メモリから２つ以上のワードを転送する。次の変形がある。

SPM{cond}{A|D}{32|16}　Rn{!}, #＜wordcount＞

SPM{cond}{A|D}{32|16}　Rn{!}, #＜wordcount＞

ＳＰＭＡに対しては、Ｐ＝０，Ｕ＝１であり、命令は次のようにエンコードされる。



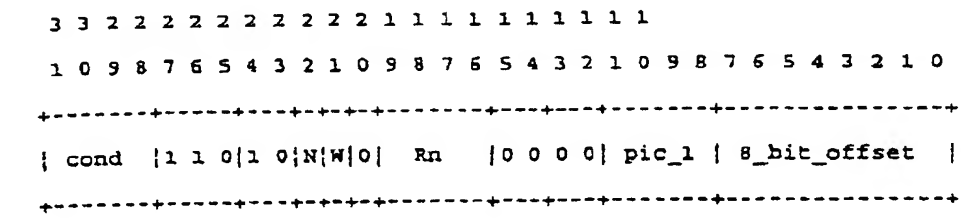
ここで、

Ｎビットは、ＳＰＭＡ３２（１）又はＳＰＭＡ１６（０）を指定する。

Ｗビットは、ベースレジスタ（１）への書き戻しを指定する。

＜８＿ｂｉｔ＿ｏｆｆｓｅｔ＞は、転送すべきデータ項目の数を指定する。

ＬＰＭＤに対しては、Ｐ＝１．Ｕ＝０であり、命令は、次のようにエンコードされる。



ここで、

Ｎビットは、ＳＰＭＤ３２（１）又はＳＰＭＤ１６（０）を指定する。

出力ＦＩＦＯからＡＲＭへのデータ転送

フォーマットの要約

ＭＰＲ命令フォーマットは、出力ＦＩＦＯからＡＲＭレジスタへデータワードを転送するためにある。ＭＰＲ命令には、次のフォーマットがある。

ＭＰＲ｛ｃｏｎｄ｝Ｒｎ

ここで、

｛｝　　は、オプションフィールドを指示する。

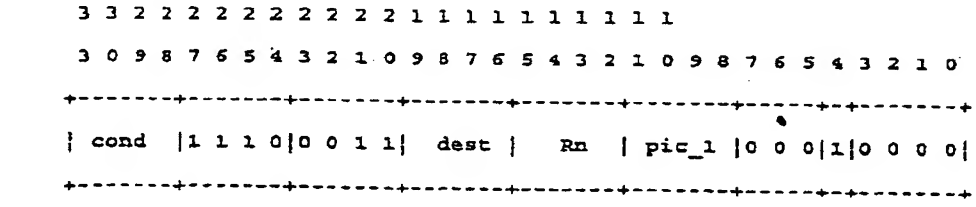
ｃｏｎｄ　　は、ＡＲＭ命令条件コードフィールドである。

Ｒｎ　　は、有効なＡＲＭレジスタ番号を求める表現である。

命令エンコーディング

ＭＲＰ命令は、ＡＲＭ　ＭＲＣ命令としてエンコードされる。

MRP{cond}　Rn



予備オプション

全体的なレベルとして、Ｐ＝０，Ｕ＝０，Ｗ＝０は、以上の形式では許可されない。将来必要になった時の予備である。

【図１】

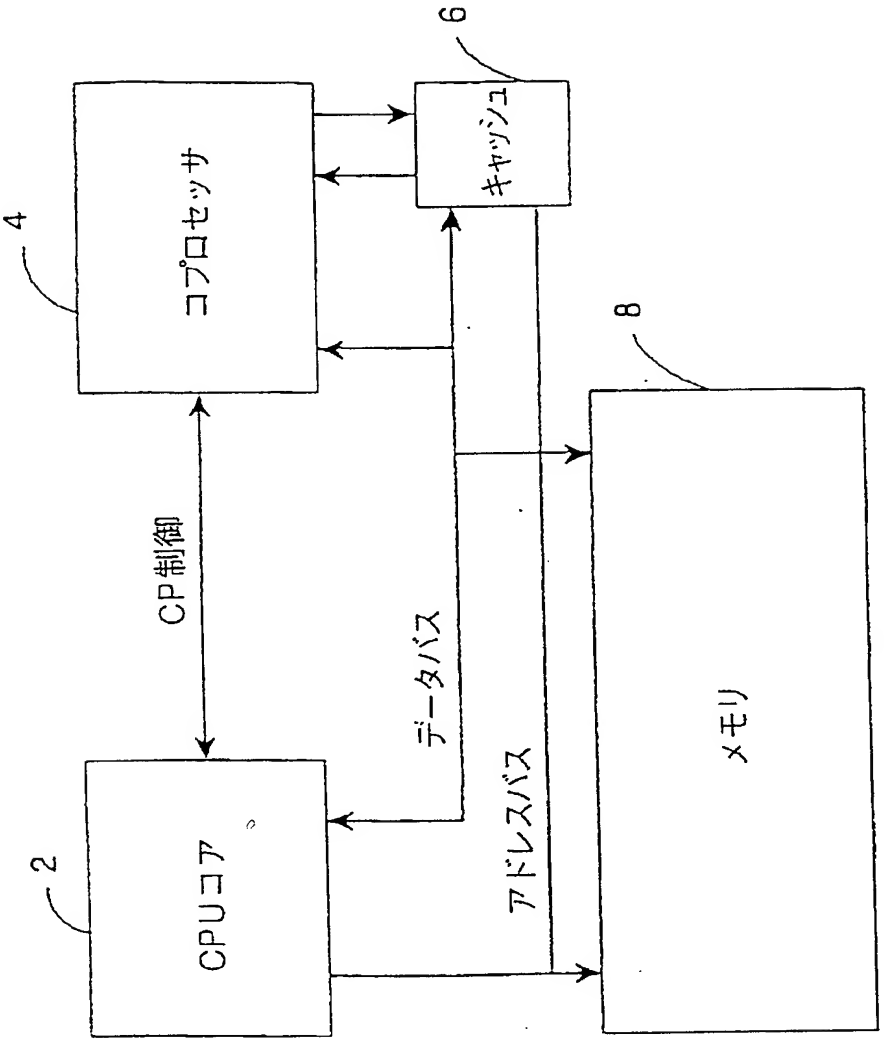


Fig.1

【図２】

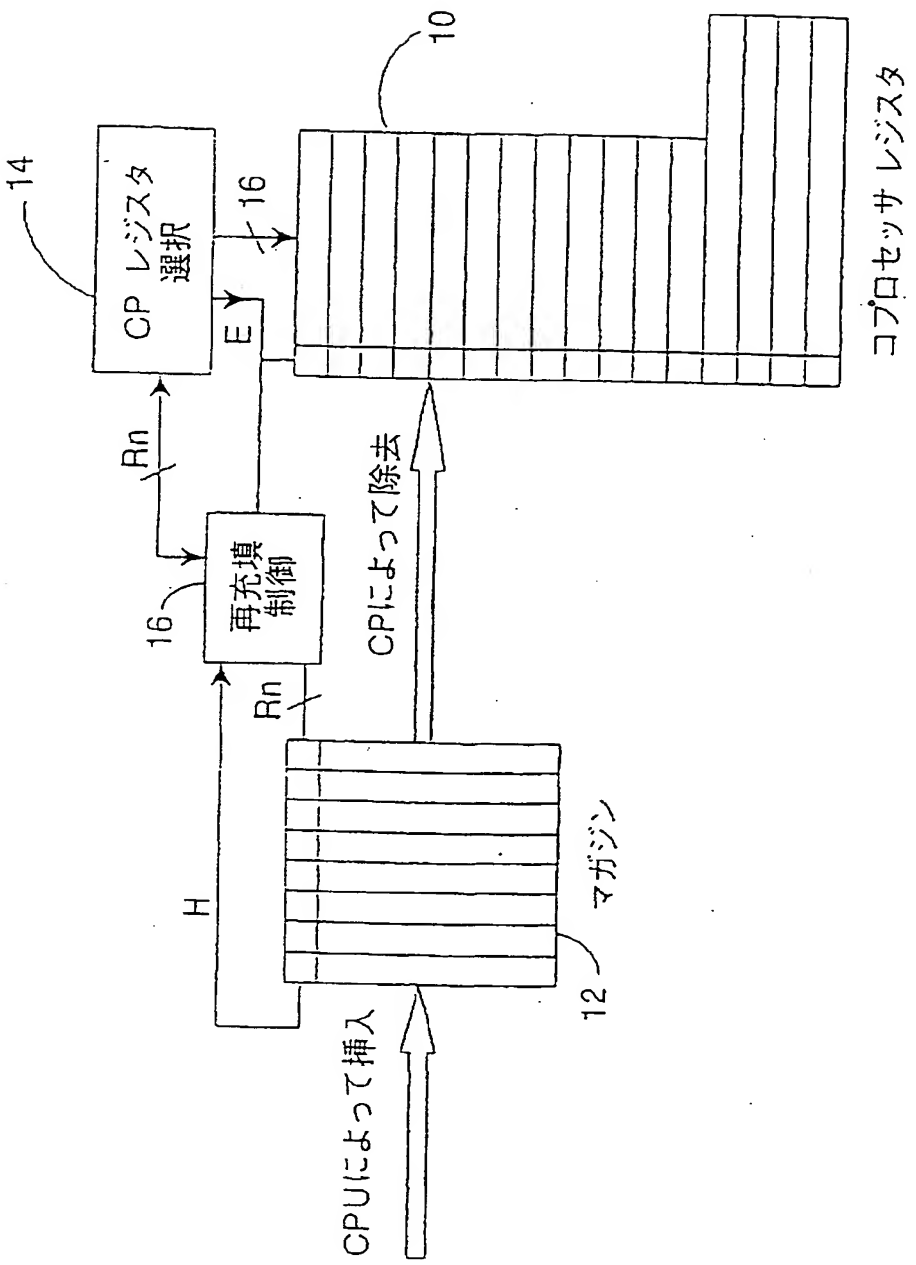

10
コプロセッサ レジスタ
14
CP レジスタ 選択
16
E
Rn
16
再充填 制御
CPによって除去
Rn
H
マガジン
12
CPUによって挿入

Fig.2

【図３】

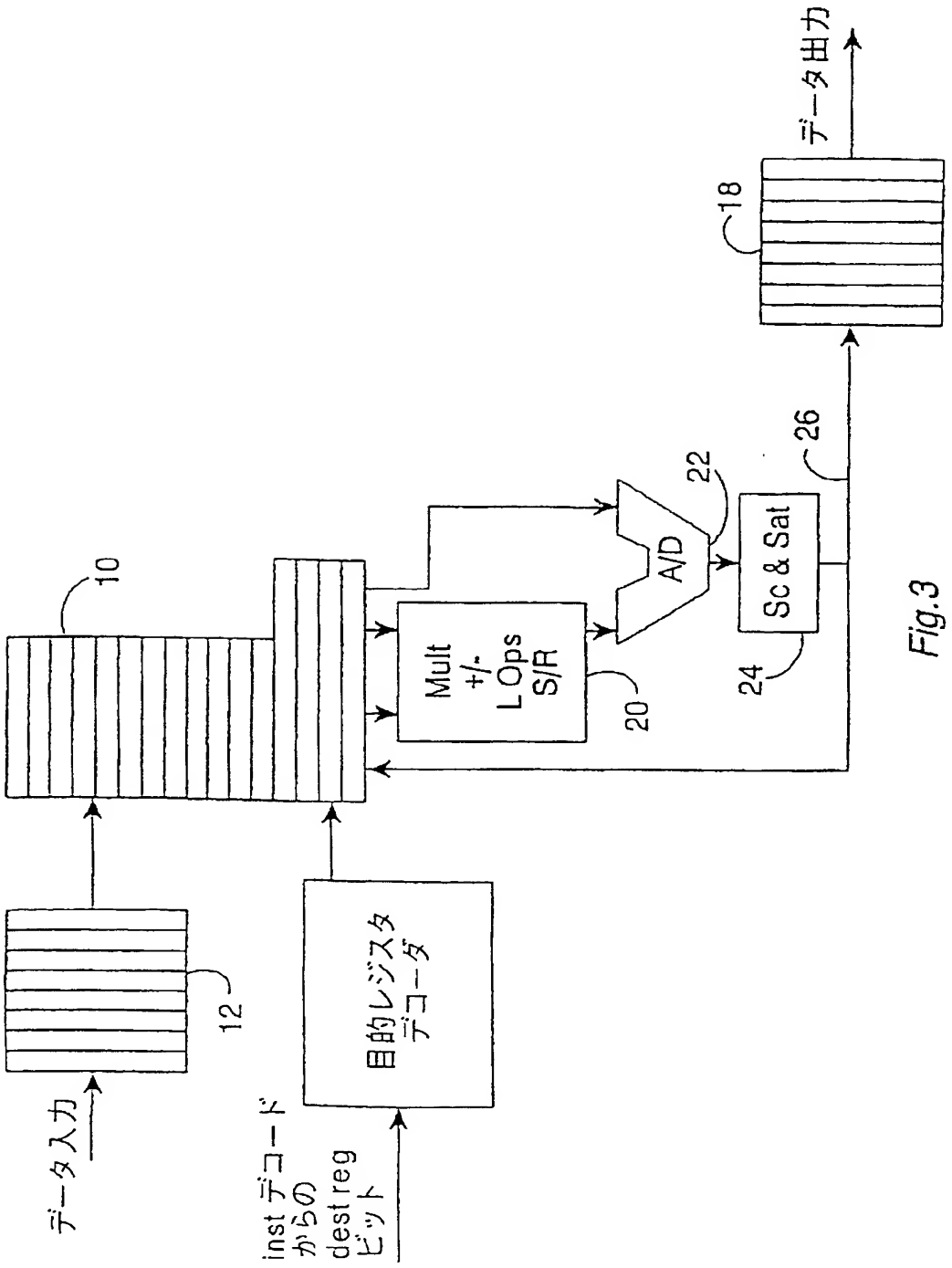

データ出力
18
26
22
A/D
Sc & Sat
24
Mult
+/-
L Ops
S/R
20
10
12
データ入力
目的レジスタ
デコーダ
inst デコード
からの
dest reg
ビット

Fig.3

【図４】

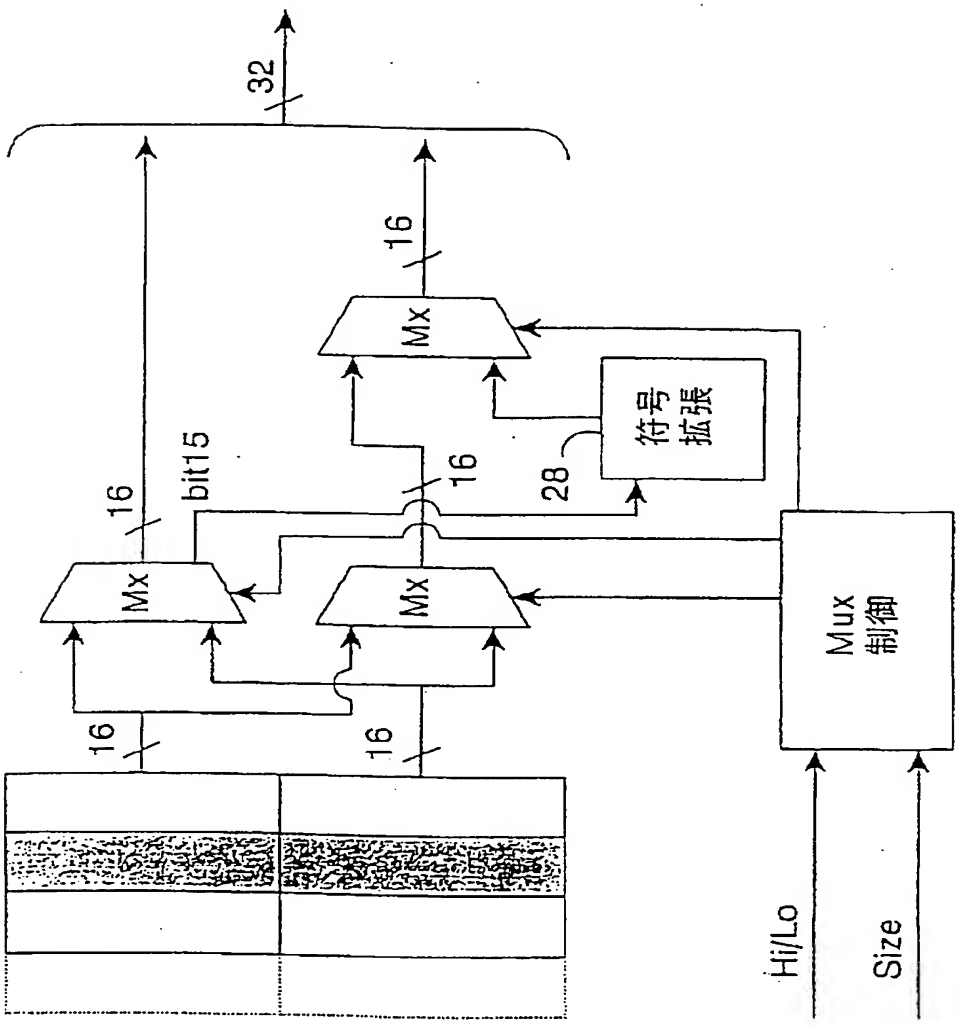


Fig.4

【図５】

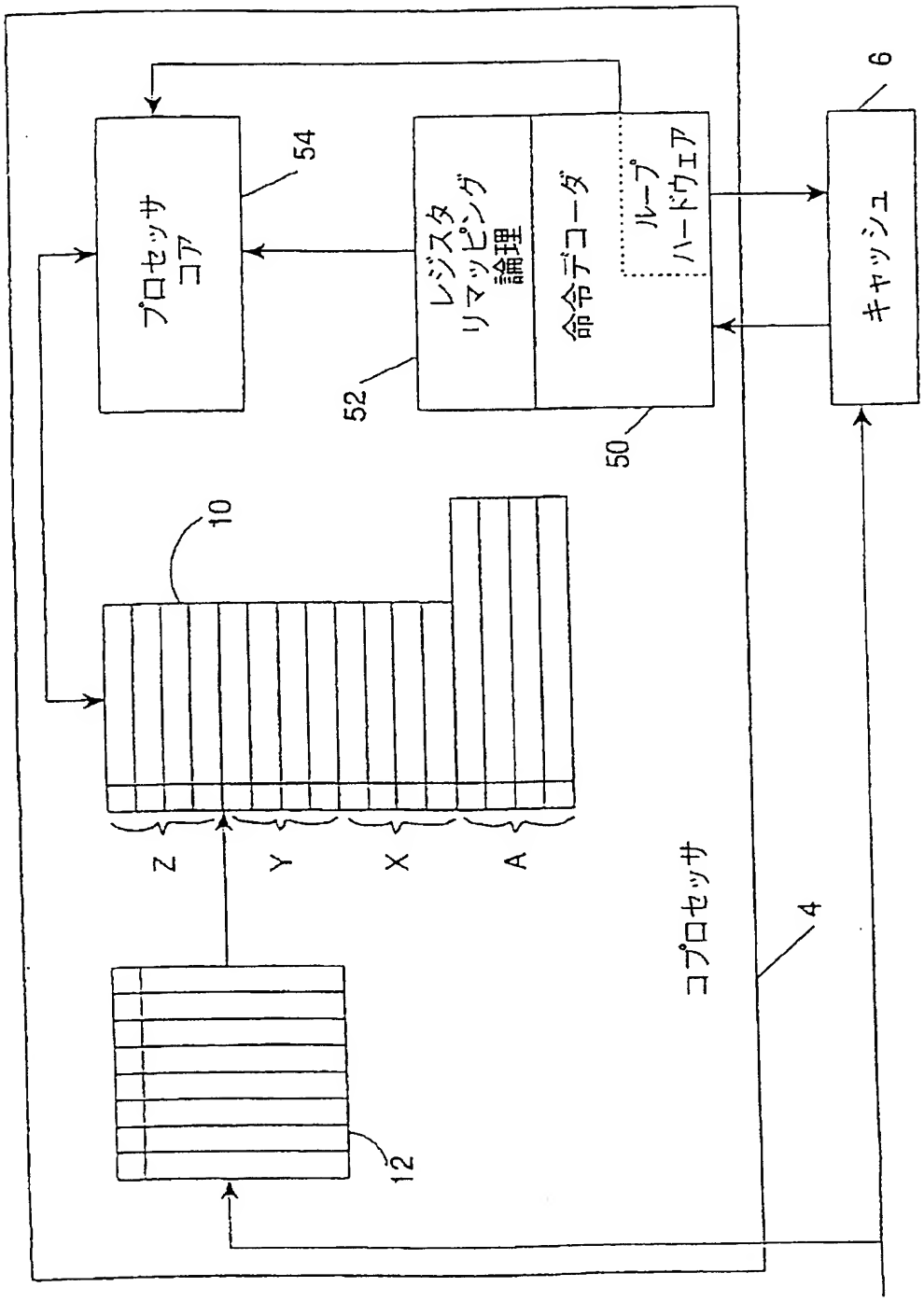

プロセッサ
コア
54
レジスタ
リマッピング
論理
命令デコーダ
ループ
ハードウェア
52
50
6
キャッシュ
10
Z
Y
X
A
12
コプロセッサ
4

Fig.5

【図６】

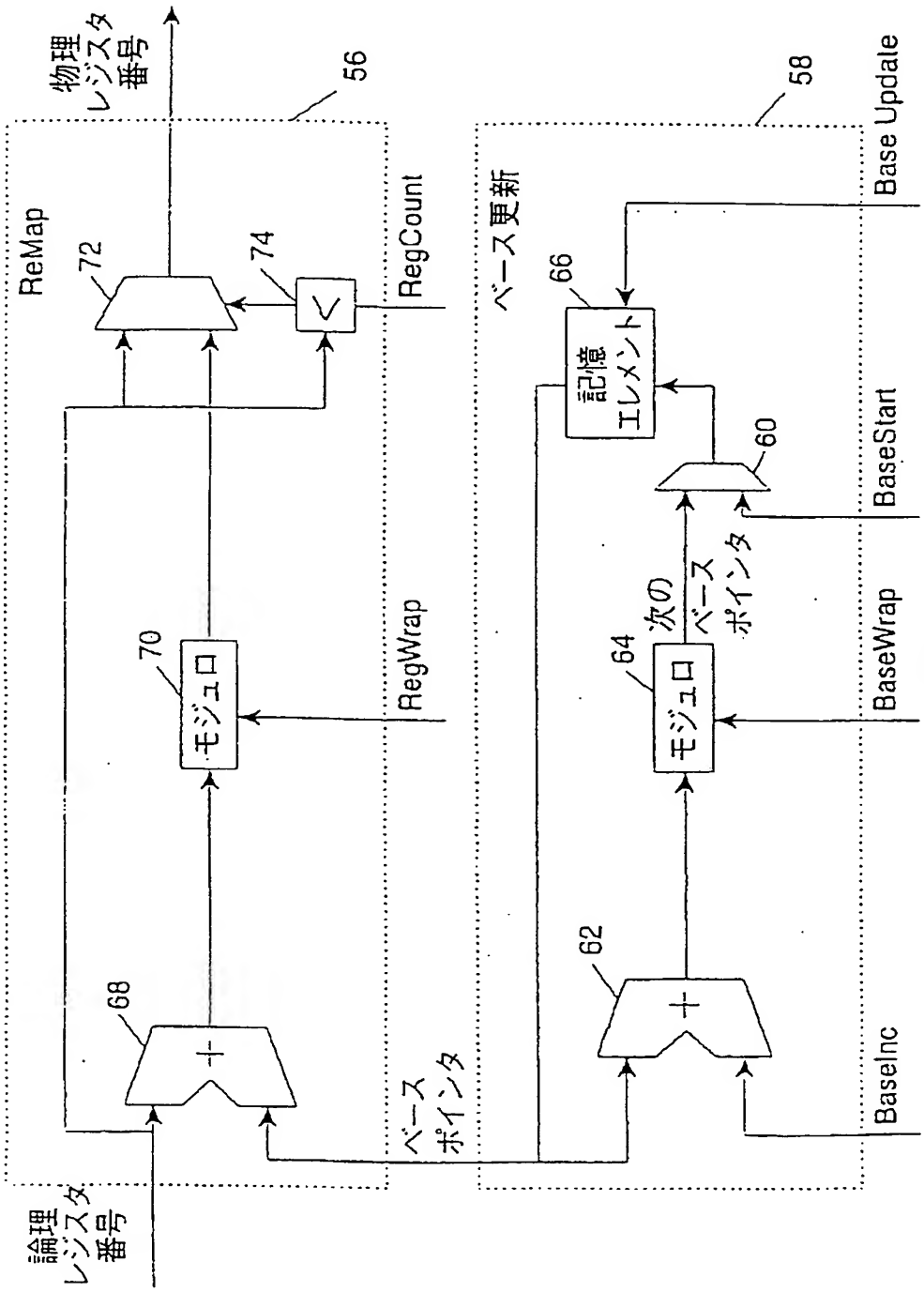

物理
レジスタ
番号
56
58
Base Updale
ReMap
72
74
RegCount
ベース更新
66
記憶
エレメント
60
BaseStart
次の
ベース
ポインタ
70
モジュロ
RegWrap
64
モジュロ
BaseWrap
68
62
ベース
ポインタ
BaseInc
論理
レジスタ
番号

Fig.6

【図７】

FIR (ブロック・フィルタ・アルゴリズム)

	d0	d1	d2	d3	d4	d5
A0=	c0	c1	c2	c3	c4	c5
A1=		c0	c1	c2	c3	c4
A2=			c0	c1	c2	c3
A3=				c0	c1	c2

Fig.7

【図８】

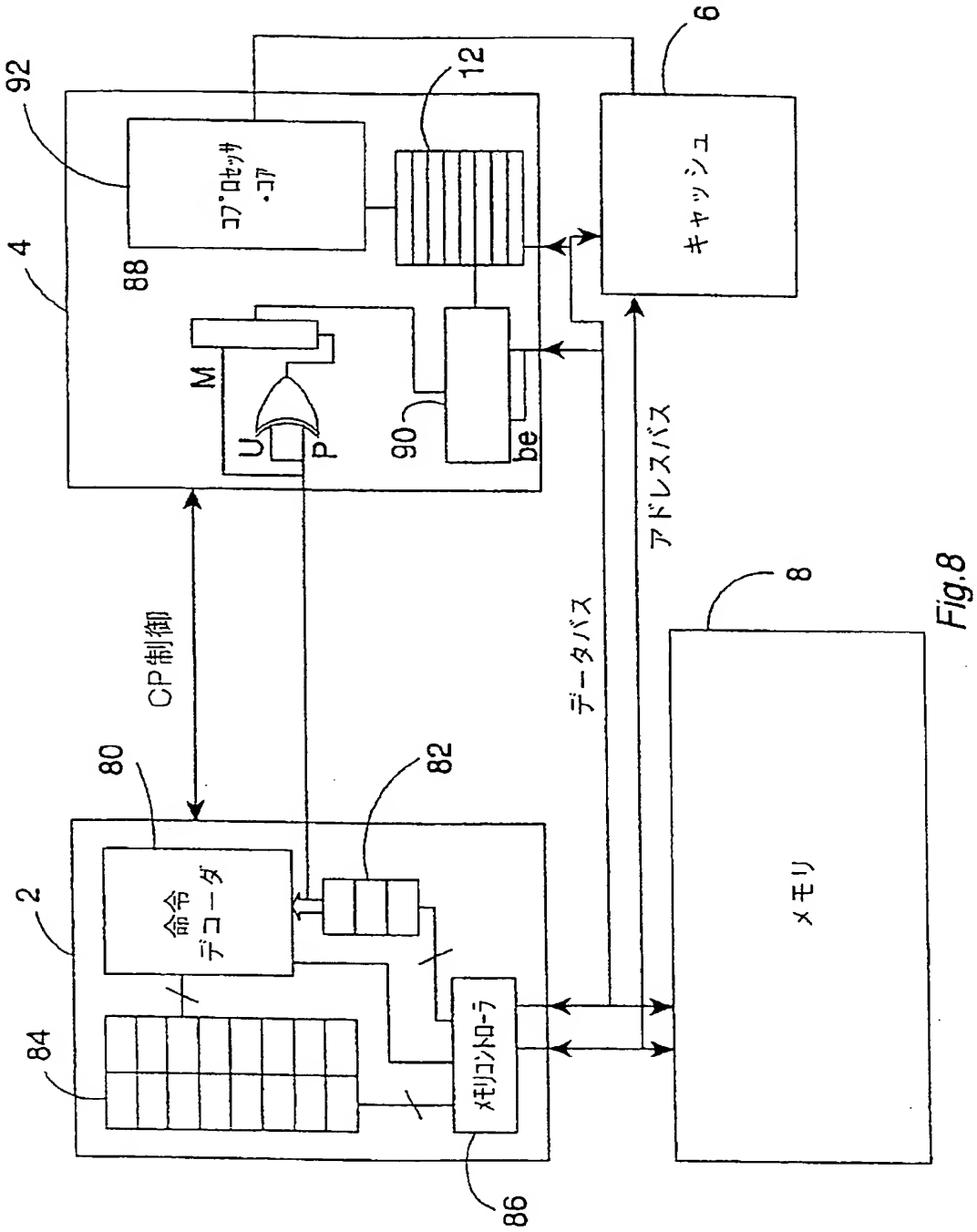

6
92
12
コアプロセッサ・コア
キャッシュ
4
88
M
U
P
90
be
アドレスバス
データバス
CP制御
8
80
82
2
命令デコーダ
メモリ
84
メモリコントローラ
86

Fig.8

【図９】

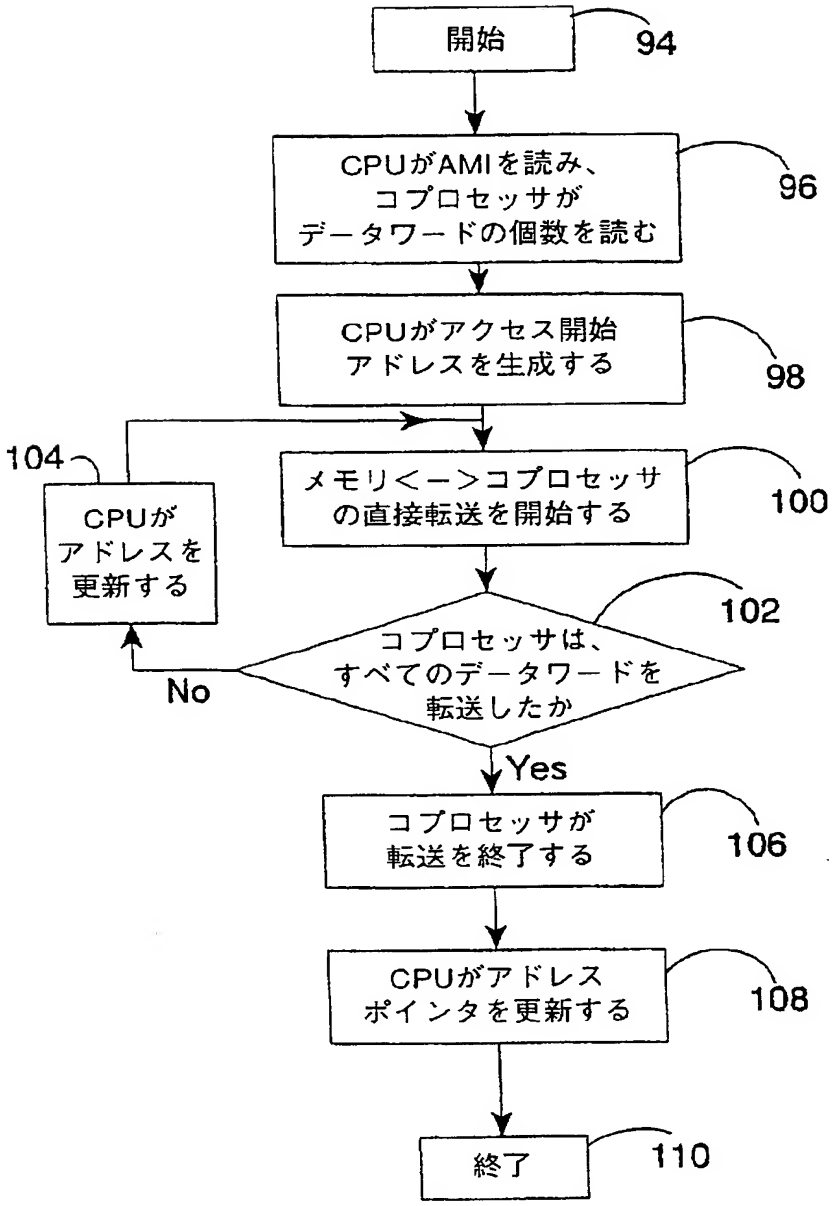

開始
94
CPUがAMIを読み、
コプロセッサが
データワードの個数を読む
96
CPUがアクセス開始
アドレスを生成する
98
104
CPUが
アドレスを
更新する
メモリ<ー>コプロセッサ
の直接転送を開始する
100
102
コプロセッサは、
すべてのデータワードを
転送したか
No
Yes
コプロセッサが
転送を終了する
106
CPUがアドレス
ポインタを更新する
108
終了
110

Fig.9

【手続補正書】特許法第１８４条の８第１項

【提出日】平成１１年４月２１日（１９９９．４．２１）

【補正内容】

（例えば、ＡＲＭ浮動小数点アクセラレータユニットにおけるように）、中央処理装置上で実行する命令内にビットフィールドを割り当てることが公知であり、そこでは、ビットフィールドがコプロセッサに渡され、転送されるべきデータワードの個数をコプロセッサに指定する。しかしながら中央処理装置上で実行される命令内で使えるビットスペースには限界があり、もし、命令内のビットがコプロセッサへのデータワードの個数を渡すことに専念すると、命令内の他のフィールド用に使えるビットスペースを制限することになる。それは例えば、命令実行に続く中央処理装置内のアドレスポインタの変更のような、データ転送に関する他パラメータを指定するのに使用されるかもしれない。

ＳＢファーバー（Ｆｕｒｂｅｒ）著「ＶＬＳＩ ＲＩＳＣ アーキテクチャ及び構築（ＶＬＳＩ ＲＩＳＣ Ａｒｃｈｉｔｅｃｔｕｒｅ ａｎｄ Ｏｒｇａｎｉｚａｔｉｏｎ）」１９８９、マルセル．デッカー（Ｍａｒｃｅｌ Ｄｅｋｋｅｒ）株式会社、ニューヨーク ＸＰ００２０６１３５８２０１９７０、ページ２６１－２６５は、中央処理装置により使用されるアドレシングモードを制御するビットと、転送されるデータワードの個数を制御するためにコプロセッサにより使用される別のビットとを備えたコプロセッサ命令を開示している。

欧州公開特許出願ＥＰ－Ａ－０，７０３，５２９は、あるオペコードによって、オフセットフィールドの一部が、使用されるオフセットよりむしろ転送されるワード数を制御するメモリアクセス命令を備えたマイクロプロセッサを開示している。

本発明の一面は、データ処理用装置であって、

コプロセッサメモリアクセス命令を含むデータ処理動作を行う中央処理装置命令を実行する中央処理装置と、

この中央処理装置に結合され、データワードを保持するメモリと、

中央処理装置とメモリに結合されたコプロセッサであって、このコプロセッサにより処理されるべきメモリ内のデータワードの指定が、中央処理装置により実

行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用して行われることを特徴とするコプロセッサと

を備え、

少なくとも１つのコプロセッサメモリアクセス命令が、前記複数のアドレシングモードのどれが中央処理装置によって使用されるかを制御するアドレシングモード情報を含み、

コプロセッサは、前記アドレシングモード情報の少なくとも一部を使用して、前記少なくとも１つのコプロセッサメモリアクセス命令に応答してメモリとコプロセッサの間で何個のデータワードが転送させるべきかを制御することを特徴とする。

本発明は、中央処理装置が複数のアドレシングモードのどれを使用するかを制御するのに使用する（レジスタ番号や即値定数を含むことのある）ビットフィールドは、コプロセッサに、転送されるべきデータワードの個数を指定するのにも使用される（命令内の他のフィールドあるいは制御レジスタに書き込まれた値のような他の因子と結合されている可能性がある）ことがある。例えば、これまでにわかっているところ、多くの場合、中央処理装置が、転送及び／又はアドレスポインタへの変更に使用されアドレスを制御するのに使用するビットフィールド情報は、コプロセッサに転送されるデータワードの個数と関係づけられるので、このビットフィールドは、コプロセッサによっても中央処理装置によっても読むことができる。コプロセッサメモリアクセス命令内で同じビットフィールドを重複して使用することにより、このようなコプロセッサメモリアクセス命令内のビットスペースを他の目的に開放することになる。更に、現在わかっているところでは、大多数の場合、転送されるべきデータワードは、少数のカテゴリに分類されるので、コプロセッサに転送されるべき個数のデータワード用に専用ビットフィールドを提供するフルフレキシビリティは、本発明を使用することによって得られるコード密度と性能の改善に負ける。

中央処理装置は、いくつかの異なる方法でアドレシングを制御できるのに対して、本発明の好ましい実施の形態においては、前記少なくとも１つのコプロセッ

サメモリアクセス命令は、アドレス値を保持する中央処理装置内のレジスタを参照し、前記アドレスモード情報は、オフセットフィールドを備え、アクセスされるべきメモリ内の開始アドレスの決定は、前記少なくとも１つのコプロセッサメモリアクセス命令の実行により、前記アドレス値と前記オフセット値の少なくとも１つから行われる。

中央処理装置のレジスタ内に保持されるアドレスポインタのそのような使用は、多大なフレキシビリティを提供し、命令内のオフセットと結合されて、アプリケーションのコーディングを簡単にする。

コプロセッサとデータアレイを介して動作することが所望される場合、本発明の好ましい実施の形態では、前記アドレス値に行われた変更が、最終アドレス値を生成し、それを前記レジスタへ記憶しなおすことができるようにする。

前記オフセットフィールドの少なくとも一部が前記コプロセッサに使用されて、データワードが何個、前記メモリと前記コプロセッサの間で転送されるかを制御できることは、非常に便利であることがわかった。

このような装置は、メモリ内に保持されているデータ上にデータ処理操作をするためにコプロセッサが使用される実際の状況のかなりの部分の要求を満たすことができる。

また、アドレシングモード情報が１つ以上のフラグを備え、前記複数のアドレシングモードのうちのどれが使用されるか、また、何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを決定するのに前記オフセットフィールドが使用されるか否かを制御できることは有利である。

アドレシングモード情報内の他のフラグビットに依存してオフセットフィールドを選択的に使用することによって、転送されるデータワードの個数をコプロセッサが制御できる方法のオプションの数を増加させ、それにより、コプロセッサメモリアクセス命令内のビット空間を余分に必要とすることなしに、実際の状況の、より高い部分の要求に合わせることができる。

高い割合の所望のタイプの動作を成就するには、好ましい実施の形態は、次のようなものでなければならない。即ち、前記コプロセッサが、前記オフセットを

使用せずに、前記メモリと前記コプロセッサの間で転送されたデータワードの個数を決定する場合、メモリとコプロセッサの間では、固定数のワードが転送される。

コプロセッサを制御するコプロセッサメモリアクセス命令モードの相補セットは、次のようなものである。即ち、前記レジスタがアドレスＲｎを記憶し、データワードがＷＬバイトの長さで、前記オフセット値がＭであり、前記１つ以上のフラグが、３つ以上の値ビットを備え、それらが、前記少なくとも１つのコプロセッサアクセス命令を選択し、以下の１つに従って操作する。

	転送開始アドレス値	アドレスレジスタ内の最終値	転送されるワード個数
(i)	Rn	Rn-(WL*M)	(固定値)
(ii)	Rn	Rn	M
(iii)	Rn	Rn+(WL*M)	M
(iv)	Rn-(WL*M)	Rn	M
(v)	Rn-(WL*M)	Rn-(WL*M)	M
(vi)	Rn+(WL*M)	Rn	(固定値)
(vii)	Rn+(WL*M)	Rn+(WL*M)	(固定値)

コプロセッサメモリアクセス命令は、以下のようなフラグを備えると有利である。

（ｉ）前記開始アドレス値が、元々前記レジスタに記憶されていたアドレス値であるか、前記オフセットフィールドによって指定される変更されたアドレス値であるかを指定するフラブビットＰ、

（ｉｉ）前記変更が、前記レジスタに元々記憶されていた値からオフセットフィールドによって指定される値を加算したものか減算したものであるかを指定するフラグビットＵ、

（ｉｉｉ）前記アドレスレジスタ内の前記最終値が、前記レジスタに記憶されな

おすべきか否かを指定するフラグビットＷ。

このようなフラグセットを使用して、コプロセッサは、高速で簡単な動作制御を行うことのできる構成となる。即ち、Ｐ　ＥＯＲ　Ｕを求めることによって、１個又はＭ個のデータワードが転送されるべきかを決定することができる。

更に、ベースレジスタがＡＲＭプログラムカウンタレジスタ（ＰＣ又はＲ１５）であれば、転送されるワード数１を生成することができる。この場合、単一ワード転送を決定するロジックが、Ｐ　ＥＯＲ（Ｕ又は（ベースレジスタがＰＣ））に変更される。

前述のコプロセッサと、中央処理装置及びメモリとの相互作用を制御する特性は多数の異なるフィールドに使用できる（例えば、浮動小数点コプロセッサ）場合、データアクセスは、次のように、比較的規則的である。即ち、前記中央処理装置と前記コプロセッサがデジタル信号処理を行い、前記メモリと前記コプロセッサの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備える実施の形態に本発明は特に適している。

本発明を他の面から見ると、本発明は以下のステップを備えるデータ処理方法を提供する。即ち、

中央処理装置により、コプロセッサメモリアクセス命令を含む中央処理装置命令を実行してデータ処理操作を行うステップと、

前記中央処理装置に結合されたメモリにおいてデータワードを保持するステップと、

前記中央処理装置に結合されたコプロセッサにより処理される前記メモリ内のデータワードをアドレスするステップであって、前記メモリは、前記中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用するステップと

を備え、

前記コプロセッサメモリアクセス命令の少なくとも１つは、前記複数のアドレシングモードのどれを中央処理装置が使用して前記メモリをアクセスするかを制御するアドレシングモード情報を含み、

前記コプロセッサは、前記少なくとも１つのコプロセッサメモリアクセス命令に応答して、前記アドレシングモード情報の少なくとも一部を使用して何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを制御することを特徴とする方法である。

本発明の実施の形態について、以下に添付図面を参照しながら、例を示す。

図１は、デジタル信号処理装置のハイレベルの構成を示し、

図２は、コプロセッサの入力バッファとレジスタ構成を示し、

図３は、コプロセッサ内のデータパス（ｄａｔａｐａｔｈ）を示し、

図４は、レジスタからハイ又はローのビットを読むためのマルチプレクシング回路を示し、

図５は、好ましい実施の形態におけるコプロセッサにより使用されるレジスタ・リマッピング・ロジックを示すブロック図であり、

図６は、図５に示されたレジスタ・リマッピング・ロジックを更に詳しく示し、

図７は、ブロック・フィルタ・アルゴリズムを示す表であり、

図８は、中央処理装置と、メモリと、コプロセッサアクセスメモリ命令を実行するためのコプロセッサとを備えたシステムを模式的に示し、

図９は、図８のシステムの動作に対応するフロー・ダイアグラムである。

以下の説明において、セクション１では、中央処理装置と、メモリと、高速デジタル信号処理能力を持つコプロセッサとを備えたシステムについて説明する。セクション２は、セクション１のシステムの変形について述べるもので、そこでは、コプロセッサメモリアクセス命令が、転送されるデータワードの個数のコプロセッサによる制御をより簡単にすべく変更されている。

請求の範囲

1. データ処理装置であって、

コプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行う中央処理装置（２）と、

前記中央処理装置に結合され、データワードを保持するメモリ（８）と、

前記中央処理装置と前記メモリに結合されたコプロセッサ（４）であって、コプロセッサにより処理されるメモリ内のデータワードをアドレスするのに、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレシングモードの１つを使用することを特徴とするコプロセッサ（４）と

を備えるデータ処理装置であって、

少なくとも１つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレシングモードのどれを使用するかを制御するアドレシングモード情報（Ｐ，Ｕ，Ｗ，Ｍ）を含み、

前記コプロセッサは、前記アドレシングモード情報の少なくとも一部（Ｐ，Ｕ，Ｍ）を使用して、前記少なくとも１つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御することを特徴とするデータ処理装置。

2. 請求項１に記載の装置であって、前記少なくとも１つのコプロセッサ・メモリアクセス命令が、アドレス値を持つ前記中央処理装置内のレジスタを参照し、前記アドレスモード情報がオフセットフィールド（Ｍ）を含み、そこでは、アクセスされるべき前記メモリ内の開始アドレスが、前記少なくとも１つのコプロセッサ・メモリアクセス命令の実行に際して、前記アドレス値と前記オフセット値の少なくとも１つから決まることを特徴とする装置。

3. 請求項２に記載の装置であって、前記アドレス値への前記変更が、最終アドレス値を生成し、それが前記レジスタに書き戻されることを特徴とする装置。

4. 請求項２と３のいずれかに記載の装置であって、前記オフセットフィールド（Ｍ）の少なくとも一部が、前記コプロセッサにより使用されて、前記メモリと前記コプロセッサとの間で転送されるデータの個数を制御することを特徴とする装置。

5. 請求項４に記載の装置であって、前記アドレシングモード情報が１つ又は２つ以上のフラグ（Ｐ，Ｕ）を持ち、それが、前記複数のアドレシングモードの

どれが使用されているかを制御し、且つ、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを決める際に、前記オフセットフィールドが前記コプロセッサによって使用されるべきかどうかを制御することを特徴とする装置。

6. 請求項５に記載の装置であって、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを決める際に、前記オフセットフィールドが前記コプロセッサによって使用されない場合、固定数のデータワードが前記メモリと前記コプロセッサとの間で転送されることを特徴とする装置。

9. 請求項８に記載の装置であって、前記コプロセッサがＰ ＥＯＲ Ｕを求めて、データワードを１個又はＭ個転送すべきかを決定することを特徴とする装置。

10. 請求項８に記載の装置であって、前記レジスタが前記中央処理装置のプログラムカウンタレジスタＰＣであり、前記コプロセッサが、転送されるデータワードが１個かＭ個かを決めるために、Ｐ ＥＯＲ（ＵＯＲ（レジスタはＰＣ））を求めることを特徴とする装置。

11. 以上の請求項のいずれかに記載の装置であって、前記中央処理装置及び前記コプロセッサが、デジタル信号処理を行い、前記メモリと前記コプロセッサとの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備えることを特徴とする装置。

12. 請求項６及び請求項７乃至１１のいずれかに記載の装置であって、前記固定数のデータワードが単一のデータワードを含む事を特徴とする装置。

13. データ処理方法であって、

中央処理装置によりコプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行うステップと、

データワードを、前記中央処理装置に結合されたメモリで保持するステップと、

前記中央処理装置と前記メモリに結合されたコプロセッサによって、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、

複数のアドレシングモードの１つを使用することによって、処理されるメモリ内のデータワードをアドレスするステップと

を備えるデータ処理方法であって、

少なくとも１つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレシングモードのどれを使用するかを制御するアドレシングモード情報を含み、

前記コプロセッサは、前記アドレシングモード情報の少なくとも一部を使用して、前記少なくとも１つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御することを特徴とするデータ処理方法。

【国際調査報告】

INTERNATIONAL SEARCH REPORT

Inter ..onal Application No
PCT/GB 98/00083

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6　G06F9/38　　G06F9/312

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6　G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	S. B. FURBER: "VLSI RISC Architecture and Organization" 1989 , MARCEL DEKKER, INC. , NEW YORK XP002061358 201970 * page 261, section 4.1.9: 'Coprocessor Data Transfer Instructions' *	1-3, 11-13 4-8
Y	EP 0 703 529 A (NIPPON ELECTRIC CO) 27 March 1996 see claims 1-5	4-8
A	FR 2 637 708 A (NIPPON ELECTRIC CO) 13 April 1990 see claim 1	4

☐ Further documents are listed in the continuation of box C.　　☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 April 1998	20/04/1998
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Daskalakis, T

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/GB 98/00083

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0703529 A	27-03-96	JP 2682469 B	26-11-97
		JP 8095780 A	12-04-96
FR 2637708 A	13-04-90	JP 2015327 C	02-02-96
		JP 2103630 A	16-04-90
		JP 7048179 B	24-05-95

【要約の続き】
合、転送されるワード数はデフォールトの１になる。